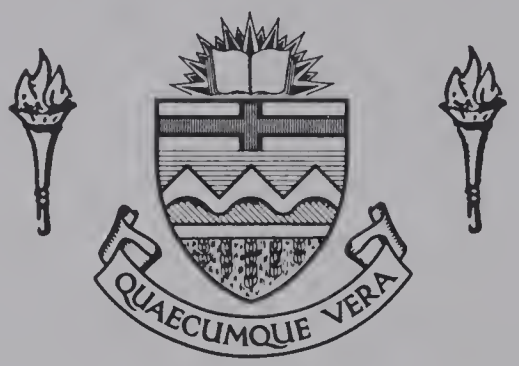
For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex libris
universitatis
albertaensis



The University of Alberta
Printing Department
Edmonton, Alberta

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Arthur R. Stanley-Jones

TITLE OF THESIS: An Inquiry Service Oriented Supervisor

DEGREE FOR WHICH THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1975

Permission is hereby granted to the UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

THE UNIVERSITY OF ALBERTA

AN INQUIRY SERVICE ORIENTED SUPERVISOR

by

Arthur R. Stanley-Jones

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL,1975

LF-153

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled An Inquiry Service Oriented Supervisor submitted by Arthur R. Stanley-Jones in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

Interactive computer access to be of maximum utility must be available for the major part of the normal work day. A problem encountered when attempting to maximize availability is a shortage of execution memory. This problem is particularily acute for systems servicing only a small number of users at a given time. Large interactive graphics services frequently service only one user at a time. Due to user reaction time the memory utilization for such specialized services is very low. As a result such services are not offered for usefully long periods of the work day.

This research analyses the problem of increasing availability by swapping interactive services in and out of the same area of execution memory. A working operating system to achieve this goal was designed and tested.

To aid in the design queuing theory and simulation were used. When a choice of simulation language was to be made it was decided to design a new interactive simulation technique based on the APL language. The new technique, CONSIM, was used to develop a working model of the operating system.

This thesis contains details on both the operating system and the simulation technique.

ACKNOWLEDGMENTS

The successful completion of this research project must be attributed to the encouragement I received from a number of people. Included in any such list must be Dr. W.S. Adams who supervised the completion of the work after my earlier supervisor moved to another institution. I also wish to thank my wife, Joannie for her patience and understanding through the complete project.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 What characterizes interactive service? 1

1.2 Why would an interactive service be requested? 4

1.3 How can an interactive service be supplied? 5

1.4 The origin of the Inquiry Service Oriented Supervisor. 7

1.5 An IP and Inquiry defined. The Inquiry Service Oriented Supervisor proposed. 13

CHAPTER 2: OTHER IMPLEMENTATIONS 15

2.1 Inquiry function on the IBM S/360 Model 20 15

2.1.1 Details of the INQUIRY function 17

2.2 An inquiry service on a CDC-3200 20

2.2.1 Details of the CDC-3200/PDP-9 system 22

2.3 LACONIQ Monitor 24

2.3.1 Details of the LACONIQ monitor 25

2.4 Comparison of the systems 26

CHAPTER 3: ISOS DESIGN 29

3.1 Operating System Structure 29

3.1.1 Operating Systems at Level One 31

3.1.2 Operating Systems at Level Two 37

3.2 ISOS Level One Conceptualization 39

3.2.1 Level I Details of ISOS Structure 43

3.3 An Instance of ISOS (ISOS at Level Two) 54

3.3.1 Level Two Details of an ISOS System 57

CHAPTER 4: SIMULATION OF ISOS 73

4.1 Why Simulate? 73

4.2 CONSIM - A Conversational Simulation Technique 76

4.2.1 CONSIM Function Descriptions 78

4.3 ISOS Simulation Results 84

CHAPTER 5: QUEUING THEORY 85

5.1 Queuing and System Design 85

CHAPTER 6: CONCLUSIONS 92

6.1 ISOS Summary 92

6.2 What next? 94

BIBLIOGRAPHY 95

APPENDIX A: ISOS Source Code 98

APPENDIX B: CONSIM Functions 115

APPENDIX C: ISOS simulator and results 122

LIST OF TABLES

Table 1.1 Usage times of S/360 8

Table 1.2 JOSS statistics 11

LIST OF FIGURES

Fig 1.1	GRID system	9
Fig 1.2	JOSS average and mean user interaction cycles	12
Fig 3.1	Users,the operating system and the hardware	32
Fig 3.2	Operating system and supervisor	34
Fig 3.3	ISOS/Host	42
Fig 3.4	ISOS priority structure	44
Fig 3.5	Worst case response	67

CHAPTER 1

INTRODUCTION

> 'If any design goal is common to all computer system organization schemes it is that of providing effective service both externally to the user of the computational facility and internally with respect to the utilization of system resources' (Sherman and Heying, 1969)

Such a goal describes a compromise demanded of a computer system designer or computing centre manager as he attempts to satisfy the usually diverse service requests of the computer user community in a resource limited environment. A request for a service with interactive response characteristics in a predominately batch processing oriented environment magnifies this conflict.

1.1 What characterizes interactive service?

After a user of a computing facility requests some service from the facility he must wait for a period of time before having his request satisfied. The delay between the request and its fulfillment constitutes response time. For terminal based requests Stimler and Bruns (1968) more precisely define response time as

> 'the time in seconds between the transmission of the last character of a message and the receipt of the first character of the reply'.

In a time-sharing system in which each user receives a time-slice or quantum of computing power according to some scheduling algorithm some authors, including Sherman and Heying (1969) define response time as the elapsed time from the end of input until the end of that request's first quantum of computing.

In an interactive environment the user of the facility attempts to utilize intermediate results from his calculations or program to modify or correct the approach for solving his problem. Frequently such a user is described as 'conversing' with his program or the computer. For an efficient 'conversation' the responses from the computer must be on a time-scale comparable to the time scale of the programmer's thoughts. The response times must, therefore, be in the order of seconds and freguently less than one second. Robert B. Miller (1968) made a number of measurements of human reactions to response times in man-computer conversational transactions. He measured seventeen types of response categories. The necessary response times ranged from Ø.1 second for control activation tasks such as turning a switch on or a light pen usage, through two to four seconds for queries regarding system status, such as, 'was the last message valid' to fifteen seconds for requests to load a program in preparation for useful work.

User response time tolerance is a function of his expectation and the state of his thoughts. If he regards a request as complex he will tolerate a longer delay before receiving an answer. The state of his thoughts, the second factor, refers to his degree of mental closure. Closure describes plateaus or completions in a group of thoughts or actions. Citing Miller's example: when telephoning another person a small closure occurs when the number is located in the directory and a second and stronger closure occurs after dialing the number. Miller (1968) stated that:

> 'The rule is that more extensive delays may be made in a conversation or transaction after a closure than in the process of obtaining closure...the greater the closure the longer the acceptable delay in preparation and receiving the next response following that closure'

He concluded:

> 'response delays of approximately 15 seconds, and certainly any delays longer than this, rule out conversational interaction between human and information systems'

Van de Goor et al (1969) in their description of TSS/8 for a PDP-8 computer consider as upper bounds on response times the interval that contains the response time to 90% of the transactions under consideration. Using these guide lines this author defines response time as the elapsed time between the transmission of the last character of the input message and the receipt of the first character of the reply and an interactive or conversational facility as one in

which the response times for 90% of the interactions does not exceed fifteen seconds.

1.2 Why would an interactive service be requested?

In the interactive or conversational mode of computing, the programmer or user may critically evaluate intermediate results and subsequently modify succeeding steps in the problem solving procedure. Conversely in the batch processing environment the user must design larger steps in his procedure, attempt the execution of the complete step, evaluate output received, usually after a significant delay (typically 2 to 30 minutes at best) and then proceed with the next step or redesign the first step if it contained errors.

Sackman (1968) summarized five studies designed to measure the relative merits of problem solving by interactive methods versus problem solving by batch procedures. A total of 212 subjects were required to solve problems covering a broad range of topics. His summary indicates that compared to batch techniques the interactive approach results in: (a) 25% less human time being used to solve the problem, (b) the elapsed time to completion of the problem is less, and (c) slightly more computer time is used. One can therefore see that for some classes of

problems the interactive approach to computing has advantages compared to batch methods. A second observation, generally true for interactive systems, is that computer processor time is spent to reduce human time and effort.

1.3 How can an interactive service be supplied?

The problem of supplying an interactive service has been resolved by a variety of means ranging from 'the service cannot be offered at this installation' to the design of total system time-sharing systems such as the Michigan Terminal System (MTS), the Xerox Batch Time-Sharing Monitor or, the Hewlett Packard Time-Sharing System.

A number of intermediate solutions in the form of sub-system monitors have also been proposed and implemented. Several specific instances of sub-system monitors are discussed in chapter two. A submonitor may be a special purpose system designed to solve a restricted class of precisely defined problems. Examples of such special purpose monitors include automated library loan services and airline reservation systems. Alternatively the submonitor may be a general purpose time-sharing supervisor. Such a supervisor allows the user to utilize several different computer languages or pre-written programs to solve many different types of problems, for example: FORTRAN could be used to solve a set of equations in engineering, a report

generator could be used for a business application, and a data base management language could be used to access a bibliography. For general purpose systems the use of the total computing system is frequently indicated. The Michigan Terminal System (MTS) represents a general purpose system. Castleman (1967) proposes a third variation, an 'evolving special purpose' time-sharing system as a combination of the above two categories. This form of submonitor allows a wide range of problem definition but only for a restricted class of applications. The common implementation of Iverson language, APL, by IBM, York University, Xerox, etc. are excellent examples of an evolving special purpose monitor. It is general purpose in that the user defines his own problems but it is restricted in that, until recently, only APL files could be accessed. Access to common system files is now possible using YORK APL file I/O or the TSIO feature of IBM's APLSV (IBM 1974). Most sub-system monitors are either special purpose or evolving special purpose supervisors.

Why are not all computing systems of the general purpose interactive variety? As mentioned earlier interactive computing trades computer time for human time. This is exactly the conflict mentioned in the opening lines of this thesis. A program in the production stage likely does not require the support of an interactive facility and

can actually execute more efficiently in the batch mode. But there are several limitations on offering both batch and interactive services. The main limitation is the lack of sufficient main memory. The on-line user separates his demands for processor time by quite long 'think periods', but the interactive program must be continually in the main memory to service the asynchronous demands for service. An amount of memory is therefore devoted to the on-line service and unavailable to batch service. This memory must be efficiently managed to be justifiable. G.H. Mealy (1966) in his description of the IBM System/360 states:

> 'no single consideration is more compelling than the need for efficient utilization of main storage'.

This thesis discusses the design of an evolving special purpose monitor which provides an interactive service for a particular class of applications. The monitor exists in a batch processing environment with a sparse resource: 'main memory'.

1.4 The origin of the Inquiry Service Oriented Supervisor.

The services offerred by the Computing Centre at the University of Alberta include an interactive graphics system and a fast turnaround student oriented batch processing facility. The graphics system, a Control Data Corporation Graphical Remote Interactive Display (GRID) system

essentially comprises a cathode ray tube controlled by a modified CDC-160A processor. The processor performs message generation, picture refreshing, and some limited calculations. Most applications require that extensive calculations and picture definition be performed by a FORTRAN program which resides in the main memory of the central computer, an IBM 360/67. The S/360 resident service routines transmit the picture definitions to the GRID system where the 160A actually controls the displaying of the required images. (Fig 1.1)

The FORTRAN programs and graphics control routines require approximately 118K bytes of high speed memory. From an examination of some typical application programs utilizing the GRID system the following characteristics emerged. Based on the elapsed time of the terminal session; (a) the FORTRAN program was idle for 97-1/2% of the time and active for 2% of the time, and (b) the control routines were handling messages for about 1/2% of the time.

Parameter	Mean (secs)	Std. Dev. (secs)
Interarrival time of requests to S/360	10.5	11.0
Message Receive Time	0.020	0.4
Service Time by FORTRAN	0.21	0.8
Message Transmit Time	0.17	0.5

Table 1.1 Usage Times of S/360 Routines for GRID Service

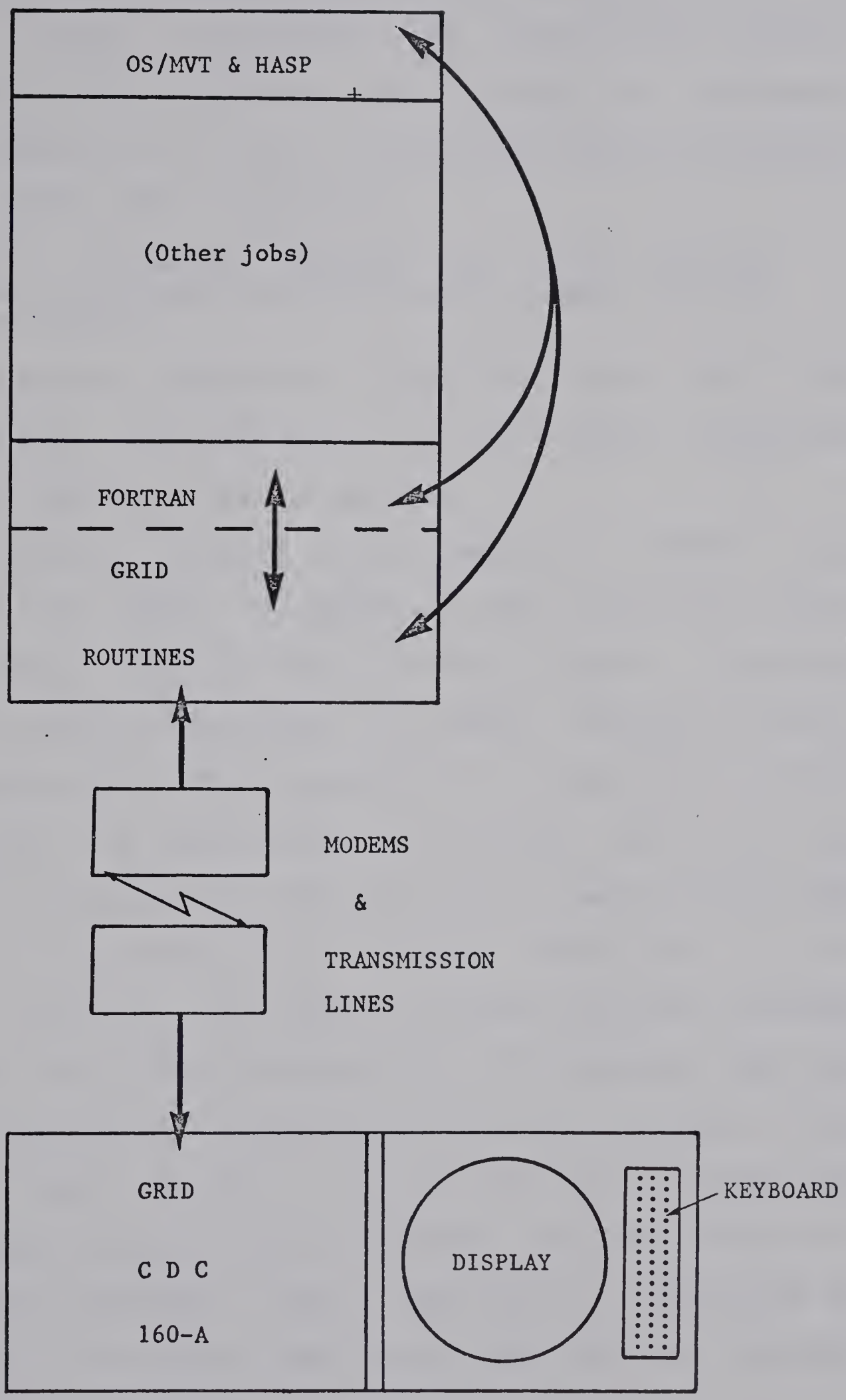


FIG. 1.1 GRID SYSTEM

These values illustrate that from the point of view of efficient memory utilization the interactive graphics application is very expensive. This example of a dedicated region is analogous to that of a dedicated machine described by Thomas (1967) when he stated

> 'with a dedicated computer the time between actions is wasted and on a large computer this is rather costly.'

The graphics application statistics show that 118K bytes of high speed memory was idle for nearly 59 minutes during each hour of graphics service.

The Student Oriented Batch Facility (SOBF) also requires 118K bytes of memory. This facility offers 'compile and go' services such as WATFOR, ALGOLW, assembler and some utility functions to enable students to solve course assignments. The programs tend to have small elapsed times in core. An examination of 46 WATFOR jobs from the 1970 Summer Session show that they have a mean elapsed time in core of 1.9 seconds with a standard deviation of 4.78 seconds. Similarly 14 student assembly language programs had a mean and standard deviation of 0.9 seconds and 0.4 seconds respectively. Assuming a normal distribution of job execution times it can be expected that 90% of WATFOR jobs will complete in less than 8.1 seconds. The SALT jobs will reduce this execution time. Therefore it would not be unreasonable to expect more than 90% of the programs

executed under control of SOBF to have an elapsed time in core of less than 8 seconds.

For further study of the characteristics of interactive applications one may consider the statistics collected by Bryan (1967) in his examination of the Rand Corporation's Johnniac Open Shop System (JOSS).

		Mean (secs)	Median (secs)
(a)	Task Turnaround	10	1.7
(b)	Compute	2	.022
(c)	Output typing	7	***
(d)	Think time	24	9.3
(e)	Interaction time	34	11

Table 1.2 JOSS statistics

From these values he produced the figure 1.2.

Bryan calculated both mean and median values because they are so different. He states that we rarely see the average case but we often see the typical or median user.

Computed on the basis of the interaction cycle time the JOSS applications had a mean and median compute time of 6% and 0.2% and an idle time of 71% and 85% respectively. These statistics and the less extensive ones this author had for graphics applications indicate that JOSS and graphics jobs have similar features.

> 'The requests that users make of the JOSS system are substantially different from those made on general purpose, on-line, time-shared system. In JOSS, there are a relatively large number of requests for short amounts of computing and a relatively small number of requests for a large amount of computing.' (Bryan 1967)

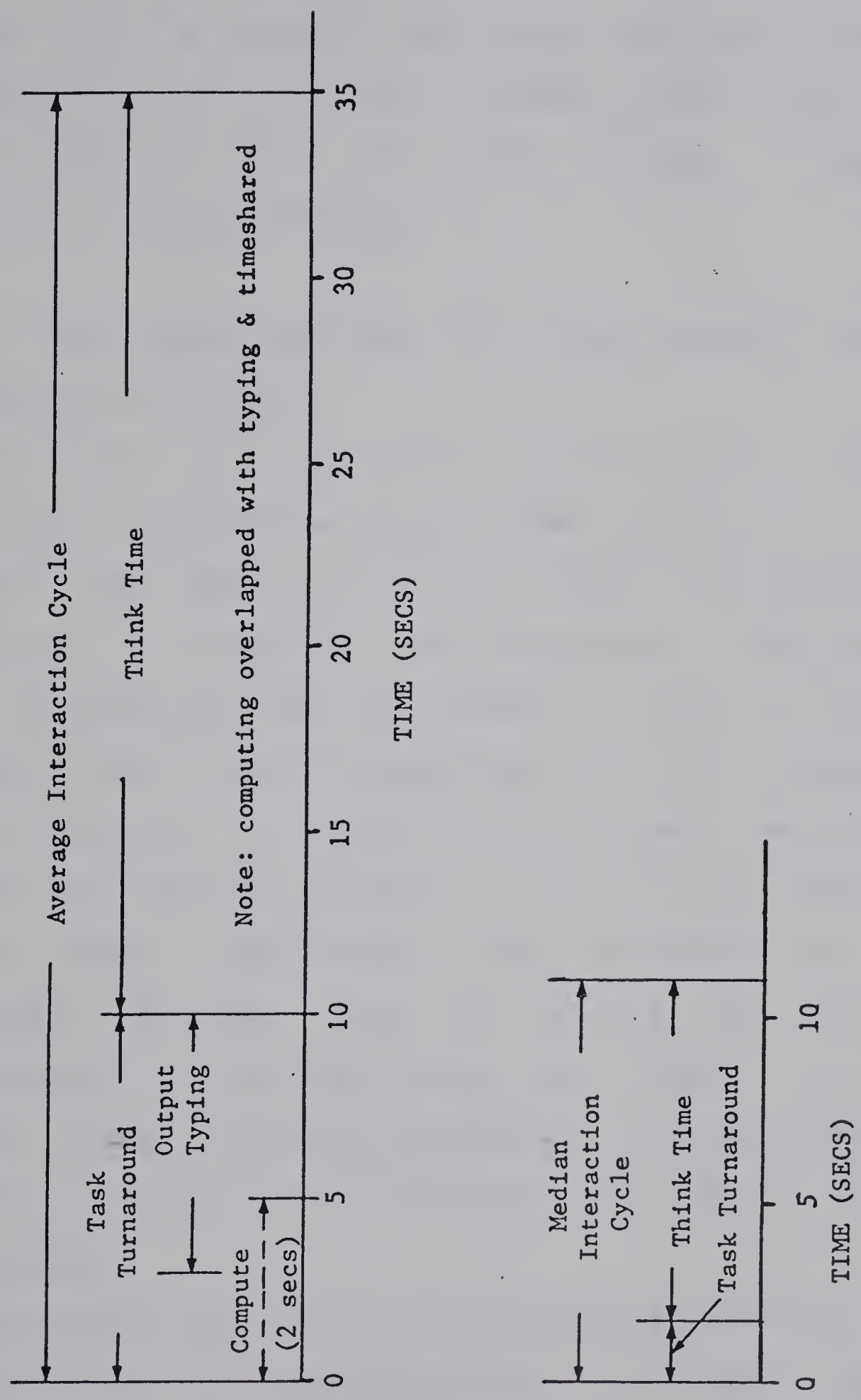


FIG. 1.2 JOSS AVERAGE & MEDIAN USER INTERACTION CYCLES

But this is exactly the case with the graphics application on the University of Alberta GRID system. With these job characteristics this author defines a class of jobs called 'inquiry programs'.

1.5 An IP and Inquiry defined. The Inquiry Service Oriented Supervisor proposed.

An inquiry program, hereafter called an IP, is defined as a computer application program whose requests for central processor time have the characteristics of an inquiry. An inquiry has two essential characteristics. Firstly the amount of processor time per request is small and secondly, the interarrival time of the service requests is large. An IP is therefore a computer job with small, well spaced, requests for processor attention. IP's are very inefficient in their memory utilization. The definition of an IP encompasses programs such as library loan services, transportation reservation systems and programs servicing requests from a satellite computer to a central computing facility. The GRID system represents an instance of the last example.

To possess real utility the services provided by an IP must be offered for extended periods of each day. However, the cost of memory to support such low utilization functions frequently precludes their being offered. For example, the

competing demands for SOBF and GRID service resulted in the low utilization graphics IP only be permitted for a few hours each day. To make IP's, such as GRID, feasible this author designed the Inquiry Service Oriented Supervisor (ISOS). This is an evolving special purpose submonitor, with capability of supporting a number of inquiry programs as foreground applications while executing batch jobs in the background. Considering the characteristics of the GRID interactions, the general response time demands of interactive services, and the execution characteristics of the SOBF programs a specific instance of ISOS was configured to support GRID as the only inquiry program and SOBF as the background application. The essential operation of ISOS comprises: (a) initiate background applications until an inquiry arrives (b) wait a period of time to allow for the possible normal termination of the background job, (c) interrupt and save the background job if necessary, (d) bring the inquiry service routines into memory (e) service the inquiry (f) save the inquiry service routines (g) restore any interrupted background jobs and (h) repeat this cycle continuously. Later chapters in this thesis will describe and evaluate ISOS in detail, particularly as it was designed to support GRID and SOBF.

CHAPTER 2

OTHER IMPLEMENTATIONS

Frequently computing bureau policy requires that batch processing utilizing a supervisor such as the IBM OS/360 or CDC SCOPE system shall be the predominant mode of operation. In such an environment, any on-line interactive services must be supported by submonitors. As the University of Alberta was predominantly a batch processing computing service the Inquiry Service Oriented Supervisor was designed. I shall dispense with any further discussions of overall time-sharing systems and concentrate on existent submonitors. Three particular subsystems, for servicing inquiry programs, IP's, as defined in Chapter 1, will be described in this chapter.

2.1 INQUIRY function on the IBM System/360 Model 20 (Darga,1970)

The IBM System/360 Model 20 Disc Programming System (DPS) supports an INQUIRY function as an option. The option was designed to permit the interruption of the currently executing program and the execution of retrievals from data files. Such retrieval requests have the characteristics of an inquiry. On the Model 20, DPS provides a single partition environment that does not support

multi-programming and therefore would normally require that the current program be aborted in order to service an inquiry. A roll-in/roll-out function which temporarily frees the main memory can be an adequate alternative to this restriction. The principle of interruption and swapping in this case developed for a one partition machine also applies to a single partition or region in a larger machine which, though it multi-programs among the partitions does not do so within them.

The INQUIRY function was initially designed as a method of permitting users of an IBM 360/20 computer to place urgent inquiries to their disc resident data files without having to cancel the currently executing program. As finally implemented the DPS INQUIRY function offers the possibility of starting any program, not just inquiry programs, without requiring that the job currently in execution be aborted. The only limitation or requirement placed on the inquiry program to be initiated is that it must have been precompiled and stored in the core image library which resides on auxiliary storage, usually a disc device. Typically inquiries processed by DPS inquiry programs request the display of one or two records from an indexed sequential data set. The keys to these records and program name are typed in by the operator. The entering of program name, file keys, and the output of results may be

overlapped with the main-line execution of the regular batch program. The actual CPU overhead for swapping, etc., to initiate an IP in a 16K byte 360/20 computer is about one second.

2.1.1 Details of the INQUIRY function

The service discipline applied by DPS to inquiries is a first-in-first-out (FIFO) regime. The operator might modify this externally by submitting a higher priority inquiry before he submits a lower priority one. The operator submits an inquiry by pressing the REQUEST key to cause an interrupt. The system v r fies that the mainline program has not prohibited inquiries. If no restriction for inquiries exists the system issues a printer-keyboard read instruction and the operator types in the IP name. The status of the mainline program is preserved, the IP is fetched from the core image library and the inquiry receives its needed service. After the inquiry has been completed the mainline program is retrieved and its execution resumes. When programs are executed in a computer the initiator routines allocate the physical resources required by the program. In the DPS - INQUIRY function the overall operating system, DPS, is not involved in the swapping and initiation of the IP´s. Therefore DPS cannot allocate the physical resources needed by the IP at the time it is

initiated. The INQUIRY function also cannot allocate resources that DPS did not allocate to it when it was initiated. To get around this problem all the resources required by permitted IP's must be allocated to the INQUIRY function submonitor. Such allocations could be made when the INQUIRY function is initiated or by any DPS controlled program that executes previously to the first IP. For example, if a particular data set, on disc, will be required by an IP, a batch run, DPS controlled, job may be executed. The batch job will define the data set in its job control statements and the DPS initiator will allocate it to that job. This job is then suspended to service the inquiries and the allocation remains effective.

Serially resuable resources, such as data files, control program routines, input/output devices, etc. pose a problem. A serious conflict might result from an inquiry attempting to use a resource, such as a printer, currently allocated to the main line program. Consider another example. The main line program attempting to update a file retrieves a record from a data set but an inquiry arrives before the updated record can be replaced. The IP updates the same record. This update is destroyed when the main line program resumes operation and completes its update function. To prevent this the monitor routines and some system data file operations are not interruptable by IP's.

Other system files and user files have a control parameter which causes the IP to abort if a conflict would result from allowing it to access the particular file.

The IBM System 360/20 has several features which distinguish it from the larger models in the 360 series. These features include (a) no supervisor/problem state differentiation and hence no supervisor call option (b) no hardware protection feature, (c) no wait state, (d) machine and program checks stop the machine and (e) no channel address word for input/output operations. To allow multiprogramming requires that the operating system schedule all requests for the use of physical devices and that one program be prevented from destroying a co-resident program. To achieve the first requirement will require that all problem programs request that the operating system schedule their requests. Such requests are made by supervisor calls. To prevent inter program destruction requires a protection scheme. As a minimum all store instructions must be checked to guarantee that they are directed to the users own memory area. Due to the large number of checks required this must be done by hardware as the overhead from software checking would be prohibitive. As neither of these requirements was met by the 360/20 the disc programming system DPS was designed as a single partition operating system.

Therefore, only a single program resides in memory at

one time. The different segments of DPS which, at a given instant, reside in core constitute the monitor. The monitor routines (a) fetch load modules from the core image library, a data set containing program segments with assigned absolute addresses, (b) perform initiation/termination procedures (c) start or queue input and output requests and (d) handle interrupts. DPS is considered to be batch oriented because any memory not utilized by the monitor is used by an application program with which the user does not converse. The monitor size i.e. the area devoted to resident control program routines, is usually about 4600 bytes with an additional 300 for the INQUIRY function. This small addition results from specifying the roll-in/roll-out routines as transient, that is, they are loaded into main memory only when needed and erased when they have served their purpose.

2.2 An inquiry service implementation on a CDC-3200 (Harrop,1970)

At the Defence Research Telecommunications Establishment, DRTE, in Ottawa a need arose for usage of a Control Data Corporation 3200 computer by remote applications. When operating under the control of the Mass Storage Operating System the CDC 3200 does not support remote service requests. An alternative operating system

and a roll-in/roll-out procedure were developed to fulfill the requirements.

At the DRTE two groups of scientists utilized PDP-9 computers and the CDC-3200 for their data analysis. The PDP-9 was used for initial data manipulation while the CDC machine was used to complete the detailed analysis. One group of scientists, carrying out ionospheric studies, punched intermediate results onto cards which were then input to the large machine. A direct link to the 3200 was desired. The second group's aim was to develop a graphics system and this demanded a direct PDP-CDC connection. The method decided upon to connect the PDP's to the CDC was to make each one look like a piece of equipment on the CDC-3200's fourth data channel. As this channel can only support eight devices the final implementation permitted up to eight remote users i.e. 8 PDP-9's to gain access to the CDC-3200.

Each user could execute precompiled stored programs without forcing the cancellation of the currently executing program. The user programs have all memory addresses assigned and are stored in a library as core images. The precompiling of the service program reduces the complexity of servicing the request and hence the response time. The elapsed time to service a remote request depends on the user application program which is invoked. The usual elapsed

time is in the order of ten seconds of which about six seconds are consumed by system overhead tasks. The ionospheric group issues requests over a period of an hour or two with an interarrival time for those requests of approximately a minute. This level of activity did not appear to seriously degrade the usual system performance. The input/output for the IP's was restricted to a dedicated disc storage area or to the PDP-9's. Therefore the high speed printers were not directly available. The severity of this restriction was reduced by making the output to the disc available to subsequent batch jobs which could process the results further or print them for the user.

2.2.1 Details of the CDC-3200/PDP-9 system

Each of the eight remote users was assigned one of the eight available equipment interrupt codes on channel four. The equipment codes on a channel are normally used to signal to the central computer that the sending device has finished servicing a request or may be used to signal a device malfunction. At the DRTE these lines were used to indicate that the particular PDP-9 wanted service. For a remote user to gain access to the CDC-3200 he sent an interrupt down his assigned line. The CDC operating system detected this as an equipment interrupt so transfers control to the 'fix-up' routine which in reality is the inquiry service submonitor.

The submonitor, called the Roll-In Operating System, (RIOS), determined which line caused the interrupt and loaded the appropriate 'error routines' which were, in fact, the stored user application programs, ie. IP's.

When a request from a remote application arrived it was initiated on a priority basis. This priority resulted from the inherent priority of the equipment codes. Upon arrival the request was held until all main program input/output activities except tape rewinds ceased. The service for the inquiry or remote request then proceeded. Other inquiries may not interrupt inquiry programs currently executing. When the remote user was serviced the interrupted program was reloaded and execution, resumed.

The CDC-3200 had one of its four channels free to service input from the PDP-9 computers which acted as remote units. A communication link and the necessary interfaces were specially constructed. The PDP-9 computers each had 8K words of memory while the CDC-3200 had 32K words. The 3200 was controlled by the Mass Storage Operatng System while executing batch programs but the special Roll-In Operatng System had responsibility for the remote requests.

2.3 LACONIQ - Laboratory computer on-line inquiry monitor (Briges,1967)

The LACONIQ monitor was developed at the Lockheed Palo Alto Research Laboratory as a submonitor to allow several users to time-share a small computer for information retrieval work. A test version not supporting any background processing was implemented on an IBM System 360 Model 30 running under the Basic Operating System (BOS). Initial results were satisfactory enough to permit an expansion of the number of terminals to be considered for the future. As for the previous two systems a roll-in procedure was used to load prestored, ready to execute application programs, IP's, as a result of a terminal request.

LACONIQ was designed to permit the apparently simultaneous use of a small computer by several programmers. The projected applications included information processing tasks such as document retrieval, update, deletion and creation. Unfortunately the files had no standard format so the monitor had to be general in design. The users of the system would not necessarily be experienced computer personnel. Bridges (1967) stated 'the basic requirement in the design of the LACONIQ monitor was that it should facilitate the programming and operation of on-line 'dialogues'. These dialogues typically consisted of

inquiries directed to a large data file. Early applications included text editing of engineering drawings, file searching, and the processing of missile component failure data.

2.3.1 Details of the LACONIQ monitor

To reduce overhead the LACONIQ dispatcher routine is event driven rather than clock driven. In a clock driven scheme the central processor is shared amongst the applications on a predetermined basis. Each application receives up to one time quantum of service then returns to the queue to await its next share of the resources. Most time-shared systems are clock driven. In an event driven regime an application program receives service until it has fulfilled its current continuous requirement. The basic timing therefore becomes a function of the completion of a quantum of work rather than a quantum of time. To guarantee satisfactory response time the designers of the LACONIQ event driven dispatcher set an upper limit on the amount of work a program could attempt at one step. If a program needed to exceed the maximum it was subdivided into program segments such that each segment was within the prescribed maximum. An event driven scheme reduces overhead in two ways. Firstly no resources are wasted in interrupting the current job when it is possible no other job is

dispatchable. Secondly, because each segment receives service to completion it need not be stored on auxiliary storage when it loses control of the processor. Therefore the monitor need only load new work i.e. roll-in but never roll-out. The LACONIQ text version on the IBM 360/30 has system overhead of less than 25%.

Bridges (1967) states:

> 'LACONIQ is programmable on any small computer that has moderate capabilities for programmed and I/O interrupts, storage protection, base registers, at least one storage disc, etc., such as an IBM 360'.

The 360/30 for the test version had 32K bytes of memory, two IBM 2311 disc drives, a data cell and a tape unit. Because the dispatcher was event driven the terminals could not expect to send data to the computer at random times but could only do so when polled by the computer. The terminals must therefore have local buffers to store input generated by the user between each polling action. The test version supported IBM 2260 and Sanders 720 cathode ray tube consoles.

2.4 Comparison of the systems

All three systems were designed to permit better than average response times for particular classes of jobs in generally 'hostile' environments. While both LACONIQ and the CDC-3200/PDP-9 system supported multiple simultaneous

users the program segmentation procedures of LACONIQ guaranteed better response time. Once an inquiry was initiated on the 360/20 Inquiry function or the CDC-3200 it might conceivably execute for minutes and effectively lock out all other inquiries. The LACONIQ designers attempted to provide two-second response times whereas the other systems were designed to reduce response time from hours or even days to minutes.

The overheads each system generated as they serviced the inquiries exhibited quite a range of values. The 360/20 Inquiry had almost no serious overhead, LACONIQ had an overhead of 25% and the CDC-3200 system had a 60% overhead. The low value for the 360/20 accrues from the fact that there was only one user and the real overhead could be overlapped with the execution of a batch program. The LACONIQ-CDC differences possibly reflect different design methods as LACONIQ supported - 3 CRT terminals and the CDC supported only 2 PDP-9's but could expand to 8. The increase in overhead with capability may not be a direct cause and effect relationship but certainly seems to be the trend in modern systems. Rosin (1967) has noted this relationship when he stated:

> 'there has been a tendency in the past few years to implement systems which are so elaborate that the amount of computer time used for solving user problems or processing commercial data can be matched or exceeded by the amount of time required by the supervisor'.

A limitation of all the systems was the requirement that all user programs be prestored in core image form. This requirement demands that the programs be developed at some other time and can not be modified or developed using one of these systems. However this author sees no reason why these systems could not be modified to allow them to be useful for program development functions. All that is required is that the service routine be a compiler and the input data be a source program. To guarantee response time, particularly on a LACONIQ type system one would want to restrict the size of the program to be compiled.

From the three systems this author particularly studied and utilized the following ideas: the problem of resource allocation and memory protection in the 360/20 INQUIRY function, the swapping techniques of the CDC-PDP se, plus the detailed state indicators, the event driver schedule and the problem state operation of the LACONIQ monitor. Hopefully some of the errors and some of the enlightment provided by these designers helped improve ISOS.

CHAPTER 3

ISOS DESIGN

This chapter will contain a theoretical overview of operating systems and a detailed description of the author's system, the Inquiry Service Oriented Supervisor (ISOS).

3.1 Operating system structure

When the design of ISOS was begun the approach tended to be completely heuristic. Certain immediate goals were known and their achievement was couched in assembly language coding. This approach contained, hopefully, much practicality but it lacked the solid theoretical foundation required to weld the individual segments into a coherent scheme. As Needham and Hartley (1969) stated:

> 'In designing an operating system one needs both theoretical insight and horse sense. Without the former one designs an ad hoc mess; without the latter one designs an elephant in best Carrara marble (white, perfect and immobile)'.

As I acquired a greater understanding of my problem the development of the theoretical structure became paramount.

Development of the theory included decisions on queue structures, service discipline, interrupt structure and even such matters as control card formats. A purely practical approach would solve, or attempt to solve, these same problems but the solutions would tend to be woven into the

very fabric of the system and would most likely lead to duplication. A better approach recognizes the problems in advance and also acknowledges that they may change in the future. The design need not attempt to solve directly all anticipated requirements but must facilitate easy modification in response to a change in the requirements. By using a modular approach sections of the system designed to solve specific tasks are clearly delineated. Each module is self contained, accepts certain forms of input and provides a result. Modules may request services from other modules. Such a structure reduces redundant coding and a given module can be modified without affecting the rest of the system. When considering overall design one must not ignore detail either -

> '... the stuff of operating system design is correct attention to detail. It could almost be said that to say a system is wrong in detail is a stronger criticism than to say it is wrong in principle ... that what distinguishes good working systems from bad ones is correct attention to detail combined with a sound underlying structure' (Needham & Hartley, 1969).

In other words the best coding cannot make a badly designed system function well but a well designed system will founder if the detail and coding are poor. As I became increasingly aware of the need for an underlying structure the design of each module, which then had a specific function, became clearer.

Earl and Bugely (1969) and Wood (1967) have published papers treating operating system structure. Earl and Bugely consider the operating system on two levels: as a concept (level one) and as an instance of that concept (level two).

3.1.1 Operating Systems at Level One

As a concept, or at level one, an operating system is that which offers the services to the end user. At level one an operating system is considered for its interrelationships with other components of the computing process rather than with specific functions or design philosophy. As a concept the operating system exists as the interface between the programmer and the hardware (Fig 3.1). In reality what many people consider to be the computer is in fact the operating system. At such a level the operating system comprises assemblers, compilers, utility routines and all those routines required to produce the environment within which programs execute. Programs comprise collections of computer instructions which perform logically complete functions. Programs can neither communicate directly nor can they invoke one and other.

The operating system divides a program into tasks, adds tasks of its own to provide the user requested service and, applies this assemblage of tasks to the hardware. From the resulting execution of these tasks the operating system

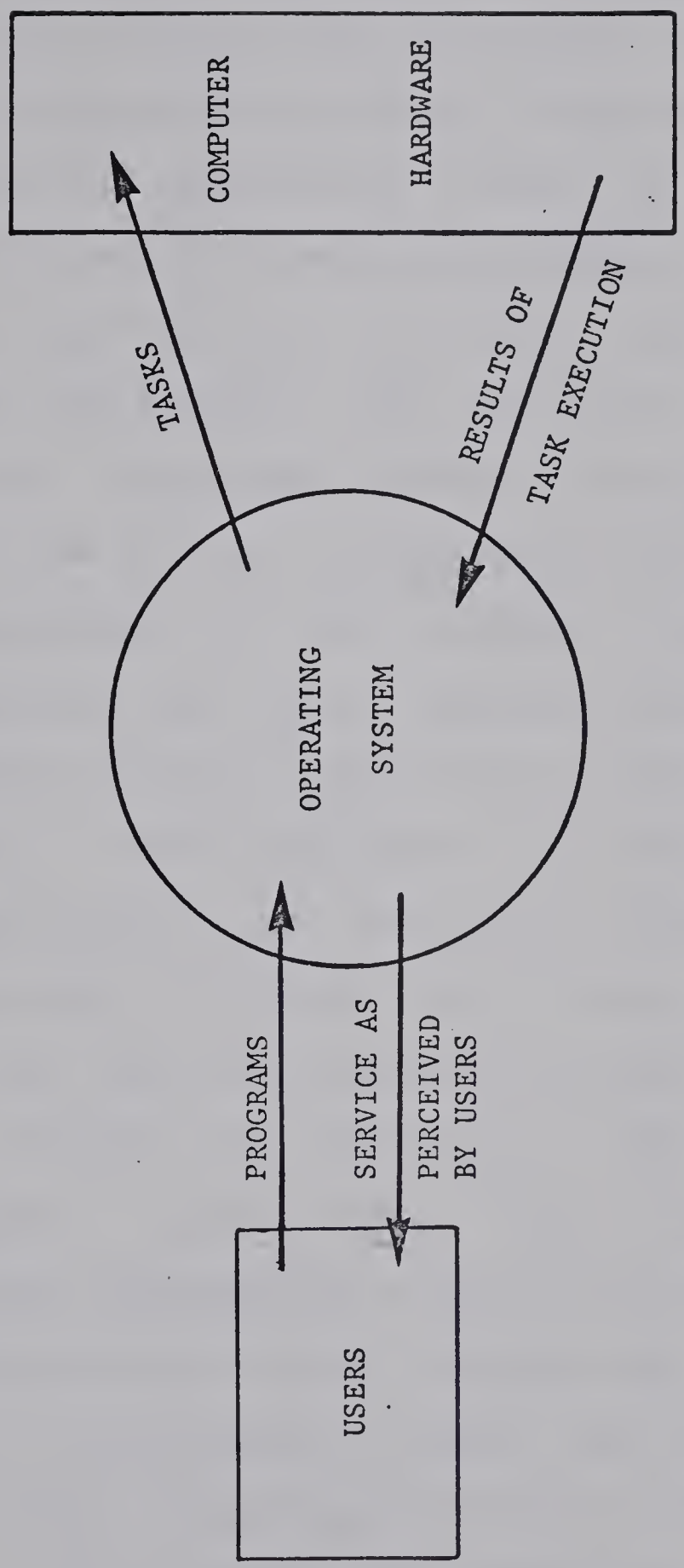


FIG 3.1 USERS, THE OPERATING SYSTEM & THE HARDWARE

presents the user with an impression and concept of the service and the computer. This level one conceptualization constitutes the main focus of Earl and Bugely. Wood extends the level one conceptualization from the operating system's relationships with the rest of the system to include the operating system's internal interrelationships. Wood (1967) identifies the supervisor as the major component of the operating system. He states: 'The supervisor exists for one purpose only; to provide service to entities known as programs'. But this is what the operating system does for users. The supervisor is that segment of the operating system which assembled the tasks, requested compilers, etc. as described above; truly the supervisor is the heart of the operating system. Wood identifies six major components within the supervisor: (a) interrupt processor, (b) input/output processor, (c) timer administrator, (d) storage administrator, (e) facilities administrator and (f) program ·dministrator. Figure 3.2 illustrates the supervisor operating system relationships and the internal component-component relationships within the supervisor.

The interrupt processor must recognize and respond to all interrupts. As mentioned in chapter two the interrupts that occur as a result of processor sharing can be generated as a function of the expiration of a quantum of time or work i.e. time driven or event driven schemes. In either case

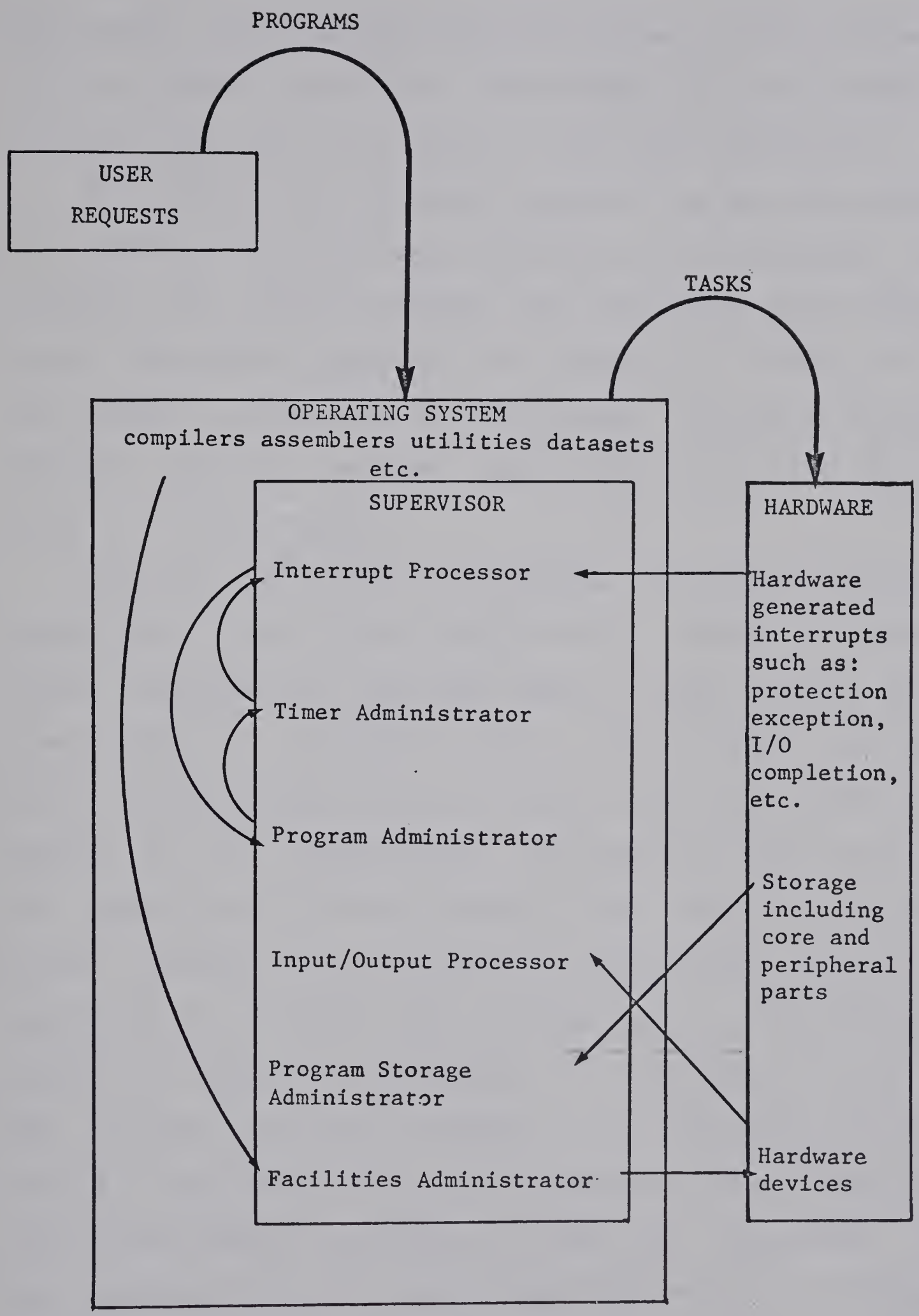


FIG. 3.2 OPERATING SYSTEM & SUPERVISOR

interrupts are generated and both schemes can be classed as interrupt driven operations. The recognition and processing of these interrupts is central to the total operation.

The input/output processor enables the operating system to communicate with hardware devices such as printers, disk storage, etc. This processor must coordinate I/O requests. Status information regarding the success or failure of an I/O request is processed by this module. Retrying of failed requests, parity checking, and error correction are all handled in this segment.

The time administrator generates interrupts to activate asynchronous tasks within the operating system or processing units. Asynchronous tasks are tasks having little or no fixed order of execution with respect to one and other. Within the operating system an asynchronous task might be a request to the input/output processor to supply the next card image from the input stream to the executing program. Another example of an asynchronous event there might be the generation of a 'twitch' of a typeball on a communications terminal to indicate that the cpu is currently servicing the last request from that terminal. In a time-sharing system which is time driven the timer is essential to indicate when time-slices expire and swapping might be necessary. The timer administrator generates interrupts which are processed by the interrupt processor. The occurrence of swapping

makes a storage administrator necessary. The storage administrator assigns execution storage (main memory) and extended storage (on peripheral devices such as disk or drum) to the programs as needed. Tables showing which programs or program segments occupy what storage are maintained as part of this storage management. Besides storage space a program will require access to data sets, teleprocessing ports, etc. The facilities administrator handles these requirements. In an IBM OS/360 system the facility administration is largely carried out by the allocation/deallocation routines in the initiator/terminator segment of the operating system.

The final segment of the supervisor, the program administrator controls the transition of the operating system between various system states and controls program priority. The available system states are supervisor state, program state and wait state. In supervisor state all interrupts are serviced and input/output is initiated. An interrupt for the supervisor might result from a program supervisor request to have an operation requiring priviledged instructions carried out. A program requests input or output but the actual data transfer is initiated and carried out in supervisor state. When a program receives control of the central processor and the instructions within the program are executed the operating

system exists in program state. Program state starts when the program administrator directs the processor to select its next instruction from the program area and it ends when the operating system reverts to supervisor state due to a supervisor call by the program or the occurrence of an interrupt. An interrupt might be caused by the expiry of a time quantum, the completion of a write operation, the receipt of an inquiry from a teleprocessing terminal, etc. When there are no outstanding requests to the supervisor and no program requires service then the operating system enters the wait state. An interrupt will occur at a future time causing a transition from wait state to supervisor state. This then is the supervisor and its functions within the operating system. The clear delineation of these operating system functions and components makes system design less ambiguous. Wood concludes his paper by stating:

> 'As systems become increasingly complex, it is increasingly important to be able to isolate the logical functions that comprise the system. -- By defining a reasonably formal structure for each function, a generalized machine - independent design emerges'. (Wood, 1967)

3.1.2 Operating Systems at Level Two

The second level of conceptualization focuses on an instance or particular implementation of an operating system. At this level we consider what services to offer

and the design philosophy as it treats topics such as queue structure, service discipline, interrupt processing, etc. Despite the fact that neither Earl and Bugely nor Wood delve deeply into level two this author feels that while level one is vital to a sound system design the general usefulness and acceptance of the system will depend on level two efforts. The user expects two things from a system (a) it works and (b) it does what he wants. Level one concepts largely guide item (a) but level two assures item (b). Level two can in some ways be equated to Needham and Hartley's 'horse sense'. For example, a beautifully implemented dedicated Algol computer would likely delight a European user but leave many North American programmers rather unimpressed. Only common sense and not theory could predict such an attitude. Therefore service selection must correspond to current and predicted requests. The queue structure and service discipline must be designed to give equitable service but also must consider machine costs and efficiency. For example, servicing all cpu requests to completion before servicing another request, i.e. no time-slicing, will reduce swapping overhead but may give unacceptable response time to subsequent requests. On the other hand the preemptive shortest job first (PSJF) discipline will give the smallest mean waiting time and the best possible service for short jobs (Coffman and Kleinrock, 1968). Unfortunately

PSJF discriminates heavily against long service requests. The design of the various queuing disciplines will be a level one task but the decision as to which to select will be a level two decision. Common sense and judgement must be used to decide on what level of service to offer each class of request.

In concluding the definition of the two levels of operating system, I define level one as - the functional design of an operating system, i.e. the purpose of its constituent parts and their interactions, and level two as - how level one is implemented in a given situation. Alternatively levels one and two can be summarized as - what the system does in general and how its done in a particular case.

3.2 ISOS Level One Conceptualization

So let us consider the Inquiry Service Oriented Supervisor (ISOS) at both levels of conceptualizaton. Firstly level one or what the system does in general. ISOS services two classes of users namely persons making inquiries via inquiry programs (I.P.'s) and a second group who do background batch processing when the system is not servicing an IP. There can be one or more IP's simultaneously under the control of ISOS. Only one IP will be receiving service at a given instant while the other IP's

will be either awaiting service or awaiting further input from the inquirer. ISOS only has one background batch job under its control at any given time. Such a background program will only receive service when no IP requires the cpu. Second and subsequent batch jobs are queued by the host computer operating system until ISOS completes the service requests of the current background job. The background jobs are batch jobs submitted by the usual batch input devices such as card readers and the output is on the usual batch output devices such as a line printer. The user has no interaction with the program between the time it is submitted and results are returned.

An IP is established by a batch job as just described but once it has been initiated the programmer interacts with it by providing further inputs, inquiries, and receives output on a conversational time-scale as discussed in an earlier chapter. The IP usually services a number of inquiries before being terminated. The inquirer uses an interactive device to supply inquiries and receive the responses. Typical devices include teletypes, IBM communication terminals, e.g. IBM-2741, and cathode ray tube devices such as Tektronix 4013. To the host computer operating system ISOS is only one of a number of programs under its control. To the IP and background programs ISOS appears as part of the operating system.

From a global point of view there is a three leveled hierarchy. At the lowest level is an IP or background program. At the middle level one finds ISOS which controls and services the programs at the lowest level but which itself is only one of many programs under the direction of the highest level, the host computer operating system. Diagrammatically this is illustrated in figure 3.3. By designing ISOS as a program to the host system it is sometimes referred to as a problem state supervisor. As such it supervises other actions and therefore has its own supervisor, problem and wait states but when it is in command of the cpu or has passed command of the cpu to one of its programs then the host operating system is in problem state. By designing the system this way the reliability is improved by using host facilities where possible, the principles of system design are more easily appreciated as the very troublesome details are handled by the host and thirdly the system is more or less portable between all computers with the same host operating system. A disadvantage is that some control functons are carried out via the host and are therefore more difficult. Therefore, when I designed ISOS as a problem state submonitor I had to follow a path similar to that which the designer of any operating system would need to follow. The host has a number of programs to control, resources to manage and tasks

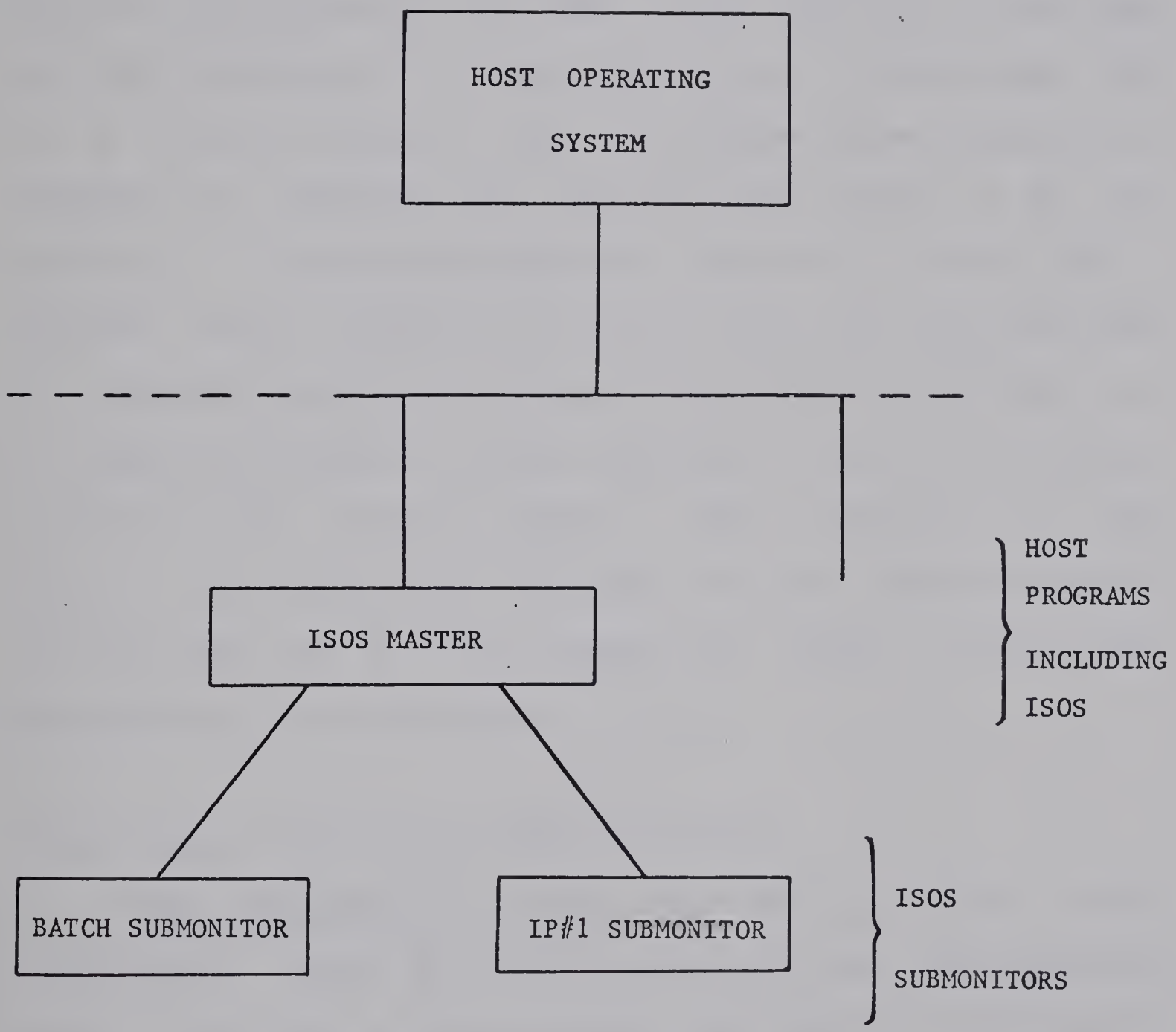


FIG. 3.3 ISOS/HOST RELATIONSHIPS

to invoke to fulfil service requests. ISOS has similar classes of resources and responsibilities. Essentially once ISOS has been initiated within the host machine a section of the host is dedicated to ISOS e.g. some execution memory is reserved exclusively for ISOS. Other sections of the host are made available on a share basis e.g. a shared data set or a utility routine. ISOS can utilize these resources as required to satisfy the demands made upon it by its programs. A conceptual difference between a host and a problem state submonitor is that to the host there are only two execution states; host state and problem state while to a submonitor there are three execution states, namely host, submonitor and program states. This difference is not severe and a submonitor designer can just consider actions by the host to be the result of 'super' instructions available to the submonitor.

3.2.1 Level 1 Details of ISOS Structure

ISOS comprises a master and two or more slave submonitors. There is a slave submonitor for the background batch jobs and a slave monitor for each class of inquiry programs. The IP slaves have a priority of i, the highest within ISOS. The master has a priority of i-1 and the batch slave submonitor has the lowest priority, i-2, (Fig 3.4).

This priority structure guarantees that an IP slave can

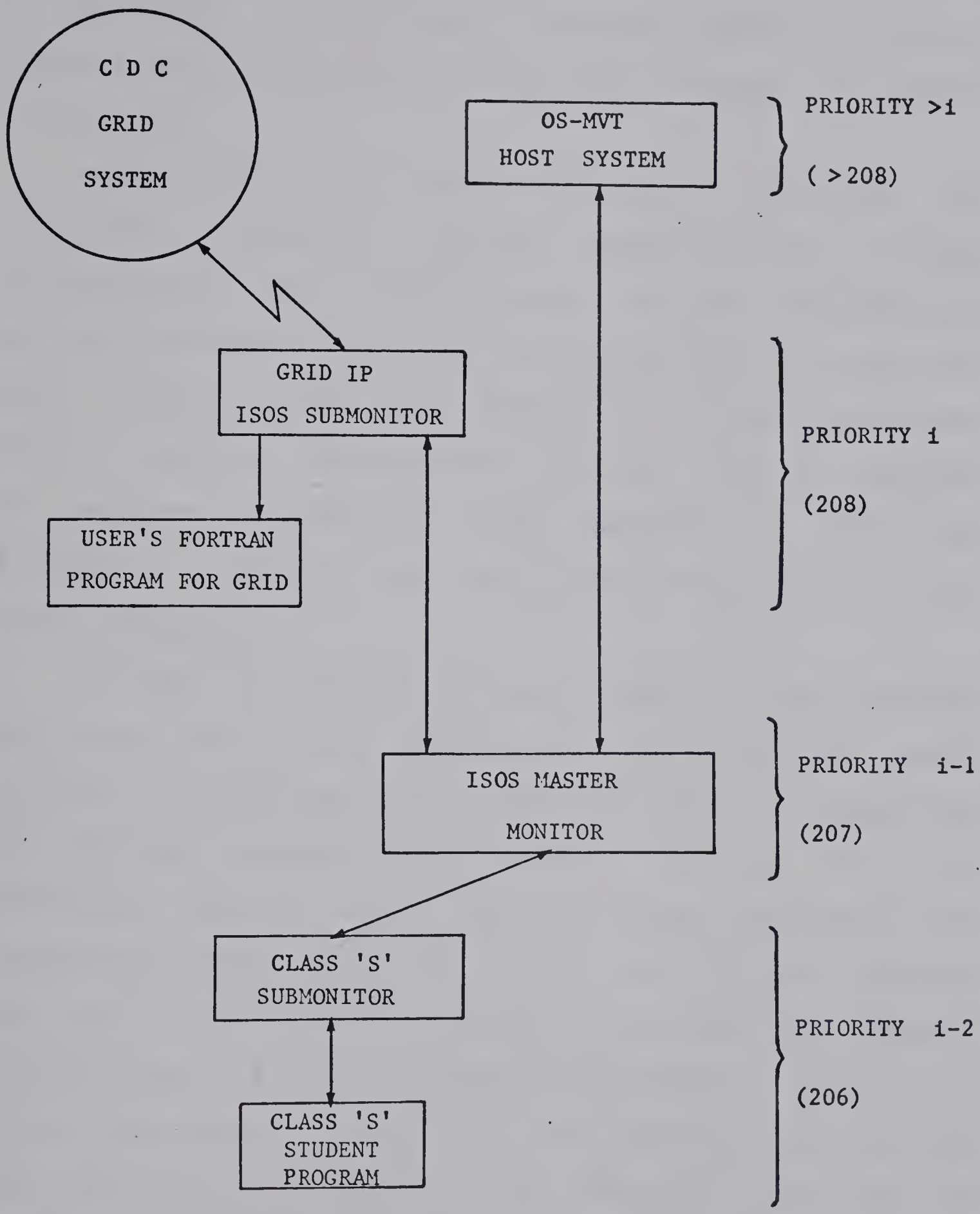


FIG. 3.4 ISOS PRIORITY STRUCTURE

preempt the cpu from the master or batch submonitors. This is necessary to permit the IP slave to briefly analyse the inquiry and notify the master that the IP needs to service an inquiry.

Wood (1967), as discussed earlier, described the interrupt processor, program administrator, storage administrator, timer administrator, facility administrator and I/O processor as the six major modules of the supervisor which itself is the key module of the operating system. ISOS has explicit coded segments for each of these modules. The program listings for ISOS (Appendix A) have been labelled to indicate the major module that the particular code implements.

In ISOS the interrupt processor module is split between the master and IP slave submonitors. There are three major sources of interrupts within ISOS that must be processed by the interrupt processor in the master : (a) an IP or IP submonitor changes state, (b) the batch program or its submonitor changes state and (c) the host system supplies the next input card as a result of an ISOS read request. Firstly there are four IP related interrupts in group (a). These interrupts include (i) a new inquiry for an IP, (ii) the completion of service of an inquiry, (iii) the IP normally completes and (iv) the IP or IP submonitor abnormally ends. Secondly, from group (b) there are two

batch related interrupts, either the batch program has normally completed or the batch program and batch submonitor have abnormally completed. The third source of interrupt occurs when a batch program has ended and the batch submonitor has become idle. If there are no pending inquiries input from the host is requested by ISOS issuing a READ request. When the READ is complete the interrupt processor receives an interrupt from the host. In all the above cases the interrupt handler must analyse the interrupt to decide to which of the 7 types of interrupts the new one belongs. Note that for each class of IP there are 4 types of interrupts. Therefore if ISOS were configured for three classes of inquiry programs there would be 15 different sources of interrupts. After identifying the source of the interrupt the interrupt processor passes control to the appropriate program administrator section to service the event.

Within the IP submonitor there is interrupt processor code that waits on a single event - the arrival of an inquiry. When an inquiry arrives this module with priority i becomes active and notifies the master that the IP needs the cpu. The submonitor then waits until the master submonitor lets it proceed.

The main function of an I/O processor is the physical input and output and the queuing of logical input/output

requests. In ISOS this is accomplished as the sole responsibility of the host. The only concern of ISOS with I/O is the verification that nothing is outstanding before a memory swap occurs and spooling IP input if a common reader is used by the master and slave submonitors. By letting the host manage almost everything relating to I/O processing the reliability of ISOS is increased while the complexity is reduced.

The timer administrator of ISOS is located partially in the batch slave submonitor and partially in the batch interrupt effector module of the master. This is the timer routine mentioned earlier. The length of the time slice, TS, is set within the time-slices. The batch interrupt effector dictates whether or not the time-slice routine of the batch slave subroutine can assign another time-slice. The size of TS will be discussed when ISOS at level 2 is discussed later in this chapter.

The program storage administrator manages both execution memory and auxiliary storage allocation. Execution memory can only be assigned to one submonitor at a time so programs of other submonitors must be removed and saved on auxiliary storage. To achieve this the storage administrator has two subsections, a core management routine to allocate execution storage and control its usage and a swap routine that manages the extended storage and directs

the actual program swapping. The storage administrator utilizes system macro instructions to manipulate memory protection keys, to determine auxiliary storage addresses and carry out the read and write requests needed for the swapping. These instructions are sufficient to indicate the validity of ISOS. For the highest efficiency these system instructions should be replaced by lower level instructions. For example in an IBM computer the execute channel program, EXCP, instructions should be used to carry out the swapping.

The facility administration involves the verification of requested batch service and the requestng from the host system needed compilers, assemblers, linkers, etc. The allocation of data sets is accomplished with job control requests to the host when ISOS is started as in the INQUIRY function in the 360/20 discussed in Chapter 2. The actual loading of the needed routines is the responsibility of the respective slave submonitor. Specifically each submonitor loads the routines requested by the current program under its control. The actual data set is allocated to ISOS by the host when ISOS is started but the routines from these data sets are loaded as directed by the submonitors.

The ISOS program administrator contains four modules: the master scheduler, the batch scheduler, the inquiry scheduler and the batch interrupt effector. The master scheduler decides the order of service to outstanding

requests for the cpu. These requests include: unserviced inquiries, interrupted batch programs, and new programs being held by the host. The batch scheduler duties are: enable the batch submonitor when a new batch job arrives, tell the batch submonitor what service the job requires, request that interrupted batch jobs be reloaded to core for continued processing, and record the end of the job status when the batch job completes. Inquiry schedulers exist for each class of inquiry that is supported by ISOS. Each inquiry scheduler performs a similar set of services for its inquiry programs. These services include (a) for new IP's any associated input is spooled to auxiliary storage for processing. (b) for new IP's and new inquiries any necessary core swaps are arranged. (c) for new inquiries the cpu is requested and if need be, interruption of the current batch job is requested, (d) enable the inquiry submonitor when the IP is in core and ready to run, and (e) processes inquiry end and/or the end of the IP.

In order to provide satisfactory response times for the inquiries it must be possible to interrupt the batch program, save its status, service the inquiry, then resume batch processing. This task was the most difficult to solve when designing ISOS. For the greatest ISOS stability the slave submonitors that control the batch jobs and IP's are subtasks attached to the master submonitor. The master is

checked out to the greatest possible degree. If the master abnormally ends then the host must restart it and all the inquiry and batch programs are lost and also need to be restarted by the user. Such a situation is not very desirable. By making each submonitor a subtask there is little chance of inquiry or batch program causing ISOS to abnormally end. At worst an IP or batch program can cause its own submonitor to also abend. But this does not affect any other program. When a submonitor, which is a subtask, abends the host transfers control to the master submonitor which has a fix up routine to automatically re-create the subtask and the submonitor is ready to process more programs. The subtask approach adds to system stability but makes interrupting the batch program more difficult. It is not just sufficient to create the batch submonitor with a lower priority because while this allows the IP's and master to gain control of the cpu it does not prevent the batch program from regaining the cpu as soon as the IP or master enter a wait state due to an event such as an I/O request. The kernel of the problem is that while the host transfers control from the lower priority subtask to the higher priority one the lower subtask remains dispatchable. What needs to be done is find a way to make the lower priority subtask non-dispatchable. This cannot be done directly within host problem state.

I solved the problem as follows. Recall that the priorities of the master, IP slave, and batch submonitors are i-1, i, and i-2 respectively and the highest priority dispatchable task gets the cpu. The main feature of my solution is time-slicing by the batch submonitor. After the expiry of each time-slice, the time-slice routine within the batch submonitor checks a flag, if it is set then the submonitor notifies the master that batch has stopped then the batch submonitor enters a wait state and is no longer dispatchable. I'll now describe the servicing of an inquiry that arrived while a batch job was executing. The batch job is executing and the interrupt handler is waiting for either the batch job to end or for an inquiry to arrive. An inquiry arrives. The appropriate inquiry submonitor with priority i compared to the batch's i-2 temporarily seizes the cpu. At this time the host system saves the needed status of the batch job which remains dispatchable and both the batch job and the inquiry submonitor are in core. The IP submonitor notifies the interrupt handler in the master that the IP needs the cpu. The IP submonitor then waits to be notified by the master that it can proceed. The master is now dispatchable and has priority i-1 compared to the batch submonitors i-2. The interrupt handler within the master decides which inquiry submonitor needs service and passes control to the appropriate inquiry scheduler. The

inquiry scheduler finds that a batch job is active. Based on a knowledge of the response time requirements of the inquiry the scheduler notifies the batch interrupt effector of the possible need to interrupt the batch job. The interrupt effector by using the response time requirements of the IP delays setting the interrupt flag for a time called IH, the interrupt hold time. During the time IH the master enters the wait state and the still dispatchable batch job continues executing. If the batch job ends within IH the interrupt effector then becomes active and signals the inquiry scheduler that it can proceed. If within time IH the batch job does not end then the interrupt effector reseizes the cpu, sets the batch 'need-to-stop' flag and releases the cpu to resume batch processing. Within the batch submonitor the time-slice size is TS seconds. Therefore every TS seconds that the submonitor is active its time-slice expires and the time-slice routine becomes active. Before the next time-slice is allocated the need to stop flag is checked. If it is set then the submonitor notifies the interrupt effector that the batch has stopped, otherwise another time-slice is allocated. Therefore within an average of TS/2 seconds of setting the 'need-to-stop' flag the interrupt effector is notified that the batch job is stopped and the inquiry scheduler is signalled that it can proceed. When the inquiry scheduler receives

confirmation that the batch submonitor is idle it requests the storage administrator to bring the IP into core from the auxiliary storage. The administrator will also save the batch program if it is only interrupted. When the IP has been loaded into execution memory the inquiry scheduler notifies the inquiry submonitor that it can proceed to process the inquiry. The delay from inquiry arrival until processing starts is discussed later in this chapter.

The inquiry service requires IS seconds. After this time the inquiry submonitor notifies the interrupt handler which passes control to the appropriate inquiry scheduler to process inquiry end status. Inquiry end may also be IP end in which case the inquiry scheduler processes end of job status to permit another IP in the same class to receive service. To complete the inquiry service sequence the inquiry scheduler passes control to the master scheduler. If no inquiries are outstanding but there is an interrupted batch program the batch scheduler is passed control. The scheduler requests that the storage administrator restore the batch job. The administrator routines also save any IP. Once the batch program is reloaded the scheduler notifies the batch submonitor that it can proceed. Another quantum, TS, is assigned to the batch program and the interrupt handler awaits either batch job end or a new inquiry; the situation at the start of the sequence described.

Wood stated earlier that the supervisor was the major component of an operating system. In an evolving special purpose monitor such as ISOS the importance of the supervisor is abundantly clear; roughly three quarters of the code is used to implement supervisor functions.

In addition to the formal supervisor sections there are (a) miscellaneous routines within ISOS to initialize and maintain itself, (b) routines to read and analyse input cards, (c) error diagnosis routines and (d) routines to list and scan cards. These utility functions don't belong to the supervisor as described by Wood but they do form part of an operating system including the ISOS operating system.

This is the ISOS concept or level one description. In essence a program state monitor to permit inquiry programs with fairly stringent response characteristics and background batch programs to be processed in the same area of execution storage.

3.3 An instance of ISOS (ISOS at Level 2)

The author generated a test instance of ISOS at the University of Alberta. The host computer was an IBM 360 Model 67 using IBM's operating system OS/MVT release 17 with HASP as the spooling package. The test case was designed to support interactive graphics and student batch processing. The interactive graphics inquiry programs, IP's, were

simulated with the characteristics of 'GRID' jobs as described in chapter 1. These graphics IP's are characterized by cpu requests of less than one second with a mean interarrival time of almost 11 seconds. The inquiries were user requests to change the picture on the display screen. The inquiry caused the IP, the user written FORTRAN GRID program, to resume computations. The computations might involve analyzing input from the GRID or card reader and, in most cases, using this information to generate a revised display picture. The revised picture was transmitted to the graphics scope for viewing then the IP enters an idle state until the next inquiry. The student batch jobs were all submitted for processing under (SOBF). As discussed in chapter 1, 90% of these student jobs had an expected elapsed processing time of less than 8 seconds. This characteristic was used to gain considerable efficiency within ISOS

In the first part of this chapter the ISOS concept and organization were described. This section's description will parallel the earlier discussion but focus on the actual test instance. A GRID - IP is established by starting the GRID hardware including the CDC-160A processor and telecommunications equipment, submitting a class G FORTRAN program, and then when the bootstrap is completed in the GRID have OS initiate the FORTRAN program. Subsequent GRID

- IP's are not passed to OS until the GRID hardware is idle. This is accomplished by coding HOLD on the JOB statement. If the user should forget to code HOLD and the ISOS would flush the job if a GRID - IP was already established. The programmer would need to resubmit the FORTRAN job. The student programs are submitted via a card reader as jobs in class S. They are processed on a first-in-first-out sequential basis. Each 'S' job is processed to completion before the next one is read for processing.

This ISOS configuration therefore has two slave submonitors one for GRID - IP's and one for class S student jobs. This structure and its associated priorities gives a good response time for inquiries as the usual programs under OS control had a priority of only 80 to 112 compared to the ISOS values of 206 and 208. Only OS gets the processor before ISOS. The coding for ISOS was all done in IBM's Assembler F language. As the test case was constructed to verify the concept and feasibility system macros such as GET/PUT or READ/WRITE were used for I/O rather the writing EXCP commands and GETMAIN/FREEMAIN were used to manipulate storage protection keys rather than using a special SVC. The ISOS master monitor size approaches 3600 bytes while each submonitor requires less than 1000 bytes. ISOS does not require a lot of main storage, the resources it is designed to save, or at least use efficiently.

3.3.1 Level 2 Details of an ISOS System

The resources needed by ISOS, or an operating system, are supplied by the facilities administrator. ISOS requires 19 data sets to support GRID jobs using FORTRAN (G), and level F linkage editor and student jobs using WATFOR, SALT, list, punch and list/punch. JCL statements processed by OS at ISOS initiation cause the allocation of these needed data sets to ISOS. AS ISOS processes jobs the GRID submonitor loads, invokes and deletes the FORTRAN compiler and the linkage editor while the SOBF submonitor manages WATFOR, SALT and the utilities. The facility administrator code segments in the master monitor and the GRID and SOBF submonitors are labelled in the code listings (Appendix A).

The I/O processor, as mentioned earlier, is quite limited. ISOS has routines for only two aspects of I/O: The SOBF submonitor purges all outstanding I/O requests before posting the 'have-stopped' flag and these purged requests are reissued when the batch processing resumes. Without this purge/restore cycle an outstanding I/O request at the time the batch job was swapped out due to an inquiry would most likely complete while the batch job was still on the auxiliary storage. When OS/MVT posted the I/O complete event control block it would destroy some IP code that could then cause an IP abnormal end. Secondly if the IP did not

abort the completion posting would not be available for the batch job so it could not resume. The second ISOS I/O concern involves the main input reader to the system. HASP spools this input to a pseudo-device from which the ISOS master and both submonitors might read in sequential first-in-first-out order. To avoid conflicts from open and closing the reader and in order that the correct record is read next when requested two procedures were developed.

Firstly the ISOS master opens the reader before using it and closes it before control is passed to either submonitor. In this way a SOBF routine can open the reader without a conflict and error condition. Secondly when a GRID job is encountered ISOS reads all its input then stores these card images on three data sets exclusively for the GRID submonitor. These three data sets contain FORTRAN source code, object modules, and data respectively. This re-spooling of the GRID input to secondary data sets correctly positions the reader so that subsequent batch jobs can be processed despite the fact that the GRID job may not, as yet, have completely processed all its input.

The interrupt processor is quite complex. When designing it one must be careful to avoid both deadlock situations where all of ISOS stops and nothing can run and race conditions in which incorrect actions are taken due to testing of state indicators that are not truly indicative of

the system state. Often the incorrect action of a race condition will lead directly to a deadlock situation. There are numerous examples of potential deadlock and race conditions within ISOS. Care must be taken to guard against these as errors are easily made. The test ISOS with only two submonitors and the master still required careful coding to co-ordinate these segments. Frequently during development the analysis of a memory dump from a failed test would reveal how a race condition had produced a deadlocked system. Consider this example. The interrupt handler may be entered from the master scheduler section of the program administrator. In the master scheduler a check is made for any outstanding inquiries or incomplete batch jobs. If none exist then a READ request is issued to OS/HASP for the next job. After issuing the READ, ISOS branches to the interrupt handler to await completion of the READ. The potential race condition exists because after the check for inquiries was made in the master scheduler a new inquiry might arrive. If the interrupt handler waited for the next job to arrive before ISOS resumed activity the inquiry service could conceivably be delayed for a long time. To avoid this the interrupt handler when it is entered always checks for outstanding inquiries and if there are none waits for either of the two events: READ complete or a new inquiry arrives. The interrupt handler also analyses a batch job normal or

abnormal end, an IP abnormal end and if there was an inquiry the handler must decode the post codes to determine which IP is required. The post codes are in the event control blocks (ecb's) which in ISOS are called Program Status words or PS's. Those associated with the first submonitor, in this case SOBF are called PSA's while those for the second are PSB's and general ones are PSG's. They are all numbered from one so ISOS code contains many references to PSA4, PSG1 or PSB2, etc. New inquiries processed by the interrupt handler are indicated by posting the INQINT ecb within handler or the INQINT2 ecb within the master scheduler. The two ecb's are needed to prevent the race condition discussed above. Both these ecb's are in the master monitor. They are posted as a result of the execution of a second segment of the interrupt processor located in the GRID submonitor. The GRID submonitor waits on the arrival of a new inquiry from the GRID user. This is indicated by OS posting an ecb against which the GRID submonitor has issued a wait. When the inquiry arrives the GRID submonitor becomes active, recall that it has the highest priority within ISOS. The GRID submonitor then posts both INQINT and INQINT2. A post code of zero indicates GRID, any subsequent inquiry submonitors would use post codes that are multiples of 4. A branch table can be used in the interrupt handler to cause the activation of the correct IP.

The program administrator section comprising the master scheduler the inquiry and batch schedulers, and the interrupt effector directs which routine to load. The master scheduler is small and easily modified. In the test system it services requests in the order: new inquiries, interrupted batch jobs, and new jobs either batch or IP (if there is not an IP of this class already being serviced.) Before scheduling any new jobs the scheduler must assure that the main memory is free. It calls the core management routine to free memory if necessary.

The batch scheduler scans the EXEC card to find the requested service. An error message results if no request is found. When a service request exists, the usual occurance, the batch submonitor becomes active and processes the job. For interrupted batch jobs the batch scheduler requests the needed memory swap then re-enables the batch submonitor. Throughout ISOS there are global status indicators such as STATE and specific general registers such as register number 4. (R4). STATE is a set of status words for each submonitor. STATE+4 is for batch, STATE+8 for the first IP (GRID in the test case), STATE+12 for the second class of IP, etc. STATE values of Ø, 4, or 8 indicate idle, interrupted or active respectively. General register 4 contain Ø, 4 or 8 to indicate a memory status of idle, batch in the memory or first IP (GRID) in the memory. The batch

scheduler or any routine which changes a submonitor state or the memory status must update the appropriate global status indicators.

The inquiry scheduler verifies that ISOS is not processing an IP of the given class when a new IP arrives at the reader. If there is such an IP the new one is flushed from the system. At the U of A this would rarely happen as the computer operator releases GRID jobs only after the user has initialized the graphics equipment. For new IP's the input is spooled and the appropriate IP submonitor enabled by posting PSx5, in the case of GRID this is PSB5. For new inquiries to existing IP's the scheduler first determines if the memory contains any active batch jobs. Such active batch jobs are halted by a call to the batch interrupt effector, (described below). If batch is idle or after it halts the IP is loaded and inquiry processing begun by re-enabling the IP submonitor. PSx2, or in the case of GRID, PSB2 controls the re-enabling of the needed inquiry submonitor. If the completion of an inquiry also signifies the completion of the IP then the inquiry scheduler resets the submonitor status to permit initiation of other IP's of the appropriate class.

The interrupt effector is quite tricky and permits an infinite variety of service response to inquiries. Given that inquiries are to receive better response time than

background jobs there are three important service disciplines available, namely: immediate preemption of the background batch job, head-of-the-line inquiry scheduling and due-date scheduling of the preemption. Immediate preemption implies that upon receipt of an inquiry the batch job is suspended and inquiry service begun. With an efficient swapping routine preemption provides the best response time to inquiries. Such good response time though, has its price; increased system overhead because the batch job requires swapping out and in each time an inquiry arrives. Head-of-the-line scheduling provides service to the inquiry when the current batch job ends and before any other batch jobs in the queue are initiated. This discipline gives low swapping overhead as only the IP needs to be swapped in or out. Because the batch job executes to completion it requires no swapping. However, because the inquiry needs to wait until the batch job ends the response time may not be satisfactory. Restrictions on the maximum processor request permitted batch jobs may make the head-of-the-line response time tolerable and the discipline worth considering.

Due-date scheduling has some of the advantages of both preemption and head-of-the-line disciplines. In the due-date situation estimates of how long it will take to service the inquiry and the required response time are used

to schedule the inquiry service at the latest possible time. In particular, the interruption of the batch job is delayed as long as possible. During this delay time the batch job may normally complete and then the IP is re-enabled and a swap of the batch job is eliminated. If the batch job does not normally complete by the required deadline then preemption is invoked to halt the batch program. Due-date scheduling reduces swapping overhead compared to immediate preemption and given response time at least equal to the design specification which the head-of-the-line method could not do.

The actual selection of which inquiry to service would most likely be made on a FIFO (first in first out) basis. All inquiries are serviced to completion. Preemption of IP's is not necessary because by definition inquiries require little cpu time and will not, therefore, seriously impact the response time of inquiries for other IP's. In the test version with only one IP class inter-inquiry interference poses no problem. In a larger system the master scheduler and interrupt handler would be modified to reflect the order of inquiry selection. Within ISOS two parameters, TS, the time-slice size in the batch submonitor, and IH, the interrupt hold time for each IP class, enable the implementor to utilize either of the three service disciplines mentioned above. As TS approaches zero all

three disciplines are possible while only head-of-the-line can exist if TS is very large. As IH becomes very small then the discipline is immediate preemption or head-of-the-line for TS small and large respectively. If IH is greater than zero then for small and large TS the disciplines are due date and head-of-the-line.

	TS very small	TS very large
IH very small (Maybe Ø)	Immediate Preemption	Head-of-the-line
IH greater than Ø	Due-date	Head-of-the-line

In the ISOS test case a due date scheme appears as the best choice. Recall the design goal of 90% of GRID response times less than 15 seconds. The mean GRID service time is Ø.21 seconds and 90% of SOBF jobs complete in less than 8 seconds. From actual tests a total memory swap requires 2.5 seconds using the READ/WRITE/NOTE macros.

What GRID response time can be expected? There are 4 cases:

1. When an inquiry arrives no batch job exists: Response time = R1 = Swap In + Service Time with an average 2.5 + .21 or 2.71 seconds.

2. A batch job is executing upon arrival of an inquiry but the batch ends within IH. the response time R2 is : $R1 < R2 \leq R1 + IH$.

3. The executing batch job does not end within IH but does end in the time-slice remaining after IH expires and the 'need-to-stop' flag is set. For the response time R3: R1 + IH < R3≤R1 + IH + TS

4. And in the worst case the batch job needs to be interrupted and swapped to disk storage. In this case the response time is R3 < R4≤IH + TS + Swap out of batch + Swap In of IP + Service Time.

Diagramatically the worst case response is represented in figure 3.5

Using the measured values and design goal in the worst case we get

R4 ≤ 15
R4 ≤ IH + TS + 2.5 + 2.5 + Ø.21
or IH + TS 9.79 seconds.

But what about system overhead besides that caused by swapping? From actual measurements the ISOS overhead is well below one second. Therefore, if IH + TS are less than 9 seconds then in the worst case response time should meet the required goal and swapping is kept to a minimum. Large TS values reduce time-slice overhead but require small IH values which cause more overhead from setting the need to stop flag.

As the mean SALT and WATFOR execution times are Ø.9 and 1.9 seconds respectively a choice of TS of 2 seconds will permit more than 1/2 the programs to complete during their initial time-slice. With TS = 2 then IH will be 7 seconds. In the worst case with a WATFOR job just starting when an inquiry arrives the designed response will only require

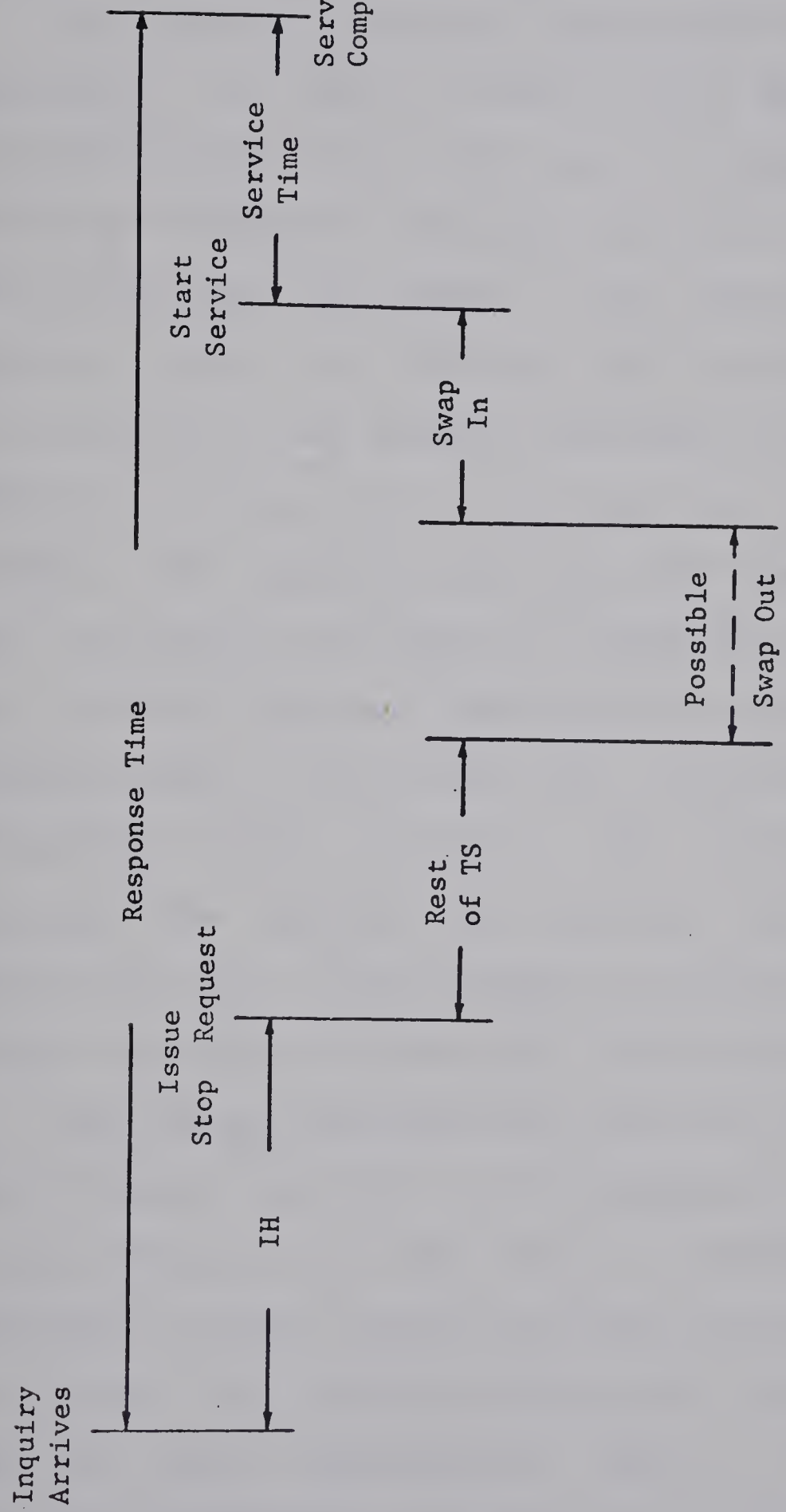


FIG. 3.5 WORST CASE RESPONSE

setting the 'need-to-stop' flag for 14% of the jobs and actually only interrupting 10% of them.

The interrupt effector exemplifies the modular coding approach. On entry to this routine general register, R1, contains the IH time for the IP class. Therefore the interrupt effector functions for all IP classes despite their differing IH values. The interrupt effector was another source of a possible race condition. The effector is activated by the inquiry scheduler if batch is busy. The effector on entry sets the IH delay by using a real time STIMER. On expiry of the delay the 'need-to-stop' flag is set. But during the delay the batch job may normally end and if the effector did not test this condition then the 'need-to-stop' would never be satisfied as the batch submonitor would be idle. ISOS deadlock results. To overcome the problem and provide the least delay to the inquiry the wait on the STIMER is actually a dual wait for either IH expiry or batch end. PSB4 indicates batch end.

The storage administrator has two large routines: the swap routine and the core management routine. The swap routine manages the swap disc or extended storage. As mentioned earlier general register R4 contains values of 0, 4 or 8 etc. to indicate main memory contents of nothing, SOBF or GRID respectively. When a scheduler needs to activate a submonitor it sets general register 7 to indicate

which routine to bring into memory; 4 for SOBF and 8 for GRID, then calls the swap routine. The swap manager compares R4 and R7 then executes any needed swaps to fulfil the request. The actual swapping utilizes BSAM READ and WRITE macros with POINT commands to locate the required code on the swapping disk. The routine determines that part of memory that the program actually uses then swaps this 'active' memory in 2Ø48 byte blocks. By not swapping the total 118K byte work area the 2.5 second swap time can be reduced.

The core management routines allocate the main or execution memory. GETMAIN and FREEMAIN macros are used to allocate and free the required areas. As the memory is already allocated to ISOS by the host operating system the GETMAIN and FREEMAIN macros are actually used to set or reset storage protection keys so the swaps, loads, etc. will not cause storage protection exceptions. When a routine is loaded the host, OS/MVT, allocates it an area of memory. An area can only be allocated to one routine, therefore if ISOS swaps the routine out then that area needs to be freed in case a future load is needed. Then when the swapped out module is returned to memory the area needs to be reallocated so that the situation is just as OS originally created it. This is especially important so that READ commands will not fail.

The final section of the ISOS supervisor, the timer administrator, is a small section that controls the time slicing in the batch submonitor. The interrupt effector decides if the batch can continue while the time-slicer allocates the time-slices of length TS. The slicer is a STIMER loop that allocates TS seconds between interrupts. At the expiry of TS seconds OS interrupts processing and transfers control to the STIMER exit routine. In the exit routine the 'need-to-stop' flag (PSA3) must be checked. If it is set then the 'have-stopped' flag (PSA4) with a code of 4 to indicate interrupted status (c.f. a Ø for job completion) is posted.

The batch submonitor then waits on the resume flag (PSA2), to continue processing. If the 'need-to-stop' was not set then the timer is reset for another TS seconds and control is returned to OS which then continues to process the batch job.

The main non-supervisor sections of ISOS perform system initialization, analyse JCL, scan input cards, list input if so required and diagnose errors issuing appropriate messages as required. The initialization routines attach each submonitor as subtasks with the correct priority. As the service programs are loaded they become part of the appropriate submonitor. Therefore, if a service program abnormally ends then the submonitor will also abend. The

service programs include SALT, WATFOR, linkage-editor and the GRID users load modules, etc. All of these routines except the GRID users load modules are quite stable. However, to prevent a subtask abend from crippling ISOS the initialization routines are designed to automatically restart any subtask that does abend. This is done by using the subtask end of task, ETX, exit facility. Within the restart routines the interrupt handler, the inquiry and batch schedulers and the interrupt effector are notified about the submonitor failure. Such notification enables ISOS to reorganize itself and reset status flags. Care must be taken in the restart routines to make sure that the subtask failure did not occur while the submonitor was being re-attached. Such an occurrence could lead to infinite loops. A control flag prevents such a loop. This is an example of the detail referred to by Needham and Hartley.

The JCL analyser uses a SCAN and LIST routine to find the ISOS JOB and EXEC statements, then lists them in the output file. The statement formats are $$ followed by an optional name, then JOB or EXEC preceeded and followed by at least one blank. On the EXEC statement the facility and service request each preceded and followed by at least one blank are coded. eg. $$TEST EXEC SOBF SALT The cards are free format in that the number of blanks is optional provided a maximum of 80 columns are used. The aim is to

prevent cancelling jobs, particularly small student jobs, due to slight errors such as those caused by fixed column formats. Any errors are clearly diagnosed by an associated error routine. The error routines are part of a general error segment that also diagnoses ISOS errors. If an ISOS error is non-recoverable then ISOS abends with a user completion code of 2000 and a SYSUDUMP.

In this section I have attempted to describe the level 2 details of ISOS, that is, how an actual system was built. The problems and pitfalls have hopefully been indicated. Appendix A contains the actual coding with the supervisor sections clearly labelled. In the next chapter a simulator is developed and used to investigate the effects of varying IH and TS.

One might ask - 'if the system is so complex is it worth producing?' James Martin provided a good answer.

> 'The advantage of time-sharing operations are proved largely with the mass of engineers, scientists and other computer users who have neither the time nor skill of a professional programmer. To these people time-sharing techniques are making computers a more practical tool for solving their problems. It has become easily accessible...' (Martin, 1967)

The main aim of ISOS was to make interactive graphics more accessible to fulfil the above prophecy.

CHAPTER 4

SIMULATION OF ISOS

4.1 Why simulate?

Simulation models are constructed despite their cost and complexity because frequently they offer the only alternative to intuition and experience. A quote from Teichroew and Lubin (1966) summarizes the situation:

> 'Mathematical analysis of complex systems is very often impossible; experimentation with actual or pilot systems is costly and time consuming, and the relevant variables are not always subject to control. Intuition and experience are often the only alternatives to computer simulation available but can be very inadequate... Simulation problems are characterized by being mathematically intractable and having resisted solution by analytic methods. The problems usually involve many variables, many parameters, functions which are not well behaved mathematically, and random variables. Thus simulation is a technique of last resort. Yet, much effort is now devoted to 'computer simulation' because it is a technique that gives answers in spite of its difficulties, costs and time required'.

John McLeod (1970) gave some good definitions of simulation and modelling:

> 'Simulation is the use of a model (not necessarily a computer model) to carry out experiments designed to reveal certain characteristics of the model and by implication of the idea, system, or situation modeled. Simuland is a term of convenience used to stand for that which is to be simulated, to avoid having to specify whether it is an idea, a real-world system, or something else.'

McLeod draws special attention to the specification of 'certain characteristics' in the definition of simulation. This distinction is vital. A model to exhibit all the characteristics of the simuland would approach the complexity of the simuland itself. It is the very fact that the pertinent characteristics can be selected that makes simulation useful. In the ISOS case I wished to investigate the effect of varying TS and IH, the time-slice and interrupt hold times, on the inquiry response time, the system overhead, and queue sizes. Other aspects, for example, the queuing of I/O requests, within the host, for swap file activity are not going to be investigated. In the ISOS simulation model swapping activity delays are simply represented by suitable random variables. To investigate the effect of more rapid swapping one need only change the appropriate random variable. McLeod draws attention to his statement in the simulation definition that the simulation experiment must be 'designed' to yield answers to the required questions. A simulator is a tool for investigating answers to specific questions it does not tell you what to do. Do not attempt to mirror the simuland in your simulator then change parameters at random to see what happens. Instead build the simplest valid model and vary the parameters in question according to a plan and measure the results on specified indicator variables. For example, one

might vary service time and measure queue length as an indicator. A second advantage of a simple model is that the simulation designer does not need an intimate detailed knowledge of all aspects of the system but rather only of his specific area. Other areas of the system are treated as appropriate random variables as mentioned above for swap time. Nielsen (1967) developed a successful model of an IBM 360/67 when other early attempts to model third generation time-shared computers were failing due to their excessive complexity. Nielsen reduced the complexity of his model by deciding to simulate the software rather than the hardware. He found that focusing on a single homogenous level simplified his model and reduced is execution time. Non software effects, eg. hardware, user behaviour etc., were simply model parameters. The deviation of those parameters might have been from another simulator but not from the one under discussion. These more simple models as well as clearly focusing on one subject are more easily modified and changed. Nielsen suggests that a designer sacrifice execution efficiency for ease of modification.

Besides investigating TS and IH I wished to investigate why many computer system programmers rarely simulate but rather trust intuition and luck. From discussions I conclude that several reasons exist: (1) They do not know a common simulation language, (2) if they know one it is

inappropriate to the task at hand, (3) if they try to use their familiar languages, FORTRAN, PL/1, etc., they get immersed in timing routines and simulation language design, (4) they find that it takes too long to build the simulator and (5) they find it hard to build a model that behaves as desired. As a means to overcome or at least reducing these problems I developed CONSIM, a conversational simulation technique.

4.2 CONSIM - A Conversational Simulation Technique.

The CONSIM technique is based on APL. In fact it is a series of APL functions that comprise a timer routine, a set of system matrices and routines to create, move, destroy, etc. entities. CONSIM has a minimum of functions but does provide the heart of a simulation language, namely, a clock for the model and a way of creating and moving entities through their simulated existence. By its very open endness a user can freely add functions for manipulating entities as he wishes without getting bogged down in the morass of timers, event chains, etc. As examples of user routines to extend the technique I have included examples of routines for adding and removing entities on FIFO and priority queues. Both of these queue types as I designed them only contain the entity ID and priority in the second case. If another user wished to place other information on the queue

he might, with a conventional language, find it to be extremely difficult to modify the queue building code and any subsequent service code. With CONSIM the user just writes the new queue routines and as the event service routines are user written they are designed for the new queue structure. For example, a queue might be modified to contain the ID plus the time the entity entered the queue. Such an addition should only be made if in the designed experiment one wished to measure the time spent in the queue.

APL is becoming increasingly widely used and even if one needs to learn it in order to use CONSIM the APL knowledge is generally useful to the user in other areas. By using APL the model can be constructed and executed interactively. The interactive construction should reduce development and modification times. The interactive execution adds a possible new feature to a model. Rather than coding all decisions within the logic of the model the person at the terminal can make the decision and indicate what action to take. This feature could prove useful in models of a system involving human input and judgment, e.g. store management situations. Using the interactive execution feature an experimenter could investigate the effect of certain decisions. With these features of CONSIM I hope to reduce the reluctance of system programmers to

simulate their problems and solutions. Appendices B and C contains a listing of the CONSIM basic functions and the CONSIM model of ISOS.

4.2.1 CONSIM Function Descriptions

In CONSIM the active elements are called entities. The names of all entity types, for example, SHIP, TRUCK, etc. are stored as rows in a literal matrix called 'ENTNAMES'. The actual entities themselves are columns in a numeric matrix named the same as the entity type. For example, if we are simulating a computing environment a type of entity might be PL/1 programs. The name used could be 'PL1JOBS' This name would be stored as a row in ENTNAMES and each program would exist as a column within the matrix 'PL1JOBS'. When an entity is processed through the simulation a vector called 'CURRENT' is created as a copy of the appropriate column of the entity matrix. The first 6 elements of CURRENT and the first rows of the entity matrix are defined as:

Element No.	Definition
1	entity number (a sequential number for all entities created in the current simulation)
2	entity type (the row number of this entity name in ENTNAME)

3	the next or current event service routine (the row number of the ESR name in ESRNAMES, see below)
4	status (see below)
5	time the entity was created
6	the next or current event time for this entity, i.e. the time at which it will be passed to the esr specified in element 3.

An entity may have any number of elements but those from 7 onwards are user specified and used. The system requires only the first six.

The entities move through the model by being processed by user written event service routines, esr's. An esr is an APL function with no explicit inputs or outputs. This function when called will need to process the entity CURRENT and before the function is exited the future disposition of CURRENT (i.e. its next esr and event time or its new status) must be specified otherwise CURRENT and the column in the appropriate entity matrix will be erased. The names of all esr's are stored as rows in the literal matrix, 'ESRNAMES'. Most of the functions in the ISOS simulation, Appendix C, are event service routines.

The main driving command in CONSIM is 'SIMULATE maxtime' - where maxtime is the value of the simuland clock at which time the simulation will halt. One required user routine is 'GENERATE'. This routine sets any variables that

the simulation needs and uses the SIGNAL command to activate the creation of initial entities for the simulation. The esr's that create entities in the model may also create signals to themselves to activate the create process at a subsequent time. Alternatively, as an entity moves through the simuland a subsequent esr may SIGNAL the necessary esr to create a new entity of the necessary type. In the ISOS simulation, 'NEWSOBF' and 'NEWGRID' create new SOBF and GRID jobs respectively and also SIGNAL themselves for future action to create another job of the appropriate type.

An event is defined as the creation of 'CURRENT' and the activation of the needed esr. The times of these events, called event times, are stored in a matrix called TIMELINE. In TIMELINE, row one is the event time, row two the sequential number of the entity and row three the entity type. The number pair, (entity number , type number), constitute the entity's ID and uniquely identify an entity. Therefore, TIMELINE actually contains the event time in row 1 and the ID in rows 2 and 3.

There are 27 functions currently contained in basic CONSIM. Of these 17 constitute basic CONSIM user commands as listed below (the actual command is in upper case and the command arguments are in lower case, the definition follows the colon).

BRANCH 'toesr': a logical branch to another esr whose literal name is the toesr. CURRENT is not passed. The command takes effect when the current esr is exited. See PASS and TRANSFER.

CONSIMLD: used to load the CONSIM system to the user workspace.

'type esr' CREATE parameters: this command creates an entity of the type named as 'type'. If this type has never existed a pattern is created and the name added to ENTNAMES. Parameter #1 must be the event time for this entity. The first esr name is passed as part of the left argument separated from type by a single blank. All other parameters are user defined. If the entity type already exists the number of parameters must match the number initially used, otherwise an error stop occurs.

DESTROY: CURRENT is replaced by 6 zeros. The appropriate column of the entity matrix is erased.

DQ 'name': provides as an output the first column of the queue specified by 'name'. See FIFOQ and PRTYQ for queue creation.

EXPON mean: provides as output a series of exponentially distributed random variables whose means were specified in the vector 'mean'.

id FIFOQ 'name': the id is placed on the end of the queue specified by 'name'. The id is concatenated as a column to the end of the queue. FIFOQ and PRTYQ are just two example of queuing that could be used. The user could write others as his own commands. Any queuing command must use the command 'QOK' to see if this queue type exists and if it does not then 'QADD' must be used to create this queue type.

MAXFREE: returns the elapsed time until the next event.

PASS 'toesr': CURRENT is passed to the esr specified by 'toesr'. The logical flow is to the end of the current esr. There is no logical branch. See BRANCH and TRANSFER.

idp PRTYQ 'name': the left argument is the entity ID and a parameter number. The ID is used to find the entity and the parameter number specifies which value is used to determine the correct position in the queue. This queue is a 3 rowed matrix with the ID in rows 1 and 2 and the actual value used to determine location in row 3. The items added to the queue are columns located such that the priority values, row 3, are in decending order. Uses both QOK and QADD. DQ returns the 3 values when used against a priority queue.

size QADD 'name': a queue called 'name' is created, its name is placed as a row in the literal matrix, 'QNAMES'. The number of rows in the queue matrix is determined by size. Usually the ID, at least, would be placed on a queue.

QOK 'name': returns a 1 if the queue called 'name' exists, otherwise a Ø is returned.

id SETSTAT s: sets the status parameter, parameter #4, of the entity identified by ID, to the value S. The status values are: Ø being processed by an esr 1 on a queue 2 destroyed (applies to current only) 3 stored (an esr has saved a copy) 4 stored (same as 3 at present) 5 on the time-line 6 user defined, in the example it was used to indicate preemption

SETTRACE ON/OFF: if tracing is set to ON the ID for current and the time is printed before each esr is called. Useful for development.

delay SIGNAL 'esr': creates a DUMMY entity with an event time of current time plus delay. This dummy will be processed by the routine specified as 'esr'. Used to cause future events and with a delay of zero can act like a Supervisor call or a POST macro.

STOP 'name': used to stop the simulation. Usually due to an error. The simulator prints the 'name' of the routine causing the halt. The simulation can be resumed, usually after error correction, by typing -> 2.

TRANSFER 'toesr': like a combined BRANCH and PASS. Both CURRENT and the logical path are diverted to the 'toesr'. Takes effect when the function is exited, therefore, is either the last line of an esr or is followed by a ->Ø statement.

These are the basic CONSIM commands. As a person develops a simulation other commands particular to the subject matter being simulated always prove to be useful. The user can write as many of these as is needed to tailor CONSIM to the particular problem at hand. In the case of ISOS, commands related to the CPU proved useful and the following were added.

CPUEND: as an esr to receive control after an EXECCPU command and schedule any queued CPU requests. Creates a new CURRENT from a copy stored by EXECCPU.

esr EXECCPU request: CURRENT is stored with parameter 3 set to point to esr, the CPU is set busy if it was idle or else the request is queued. The request can be either a single value for an amount of CPU time or a number pair in which the first element is the total request and the second element is the quantum size for requests. At the expiry of the utilization, CPUEND is activated.

PREEMPT id: the entity specified by ID is preempted from the CPU. The remaining request is saved.

RETURN id: the entity preempted is returned to control of the CPU for the time remaining when it was preempted.

These are the commands for CONSIM and the technique for adding new commands. The other functions in the CONSIM listing are simply those used to implement these commands.

4.3 ISOS Simulation Results

The ISOS simulator comprises 27 functions to model the segments of ISOS. In addition, there were the new commands mentioned earlier and a few miscellaneous housekeeping functions. GENERATE was designed to ask for inputs of such values as time-slice size, interrupt hold time, number of SOBF jobs per hour, etc. The simulator was run long enough to test all functions and illustrate its use. A sample run with TRACE is reproduced in Appendix C.

To make any simulation useful one must design experiments and make measurements. In the ISOS case, IH and TS were to be varied then response and batch job delay could be analysed. To do this several variables, e.g. RESPONSE, BREQ(batch request) and BTIME (batch elapsed time) were created and updated by the appropriate esr. At the end of the simultion a statistical analysis routine can be used to examine these values. Included with the sample run is the use of DSTAT from STATPACK II (Smillie,1966) to analyse the above mentioned variables.

The simulation was very useful and given a needed ISOS application it could be run to provide a good guide to ISOS operation in a particular environment. In this thesis it was used to verify the validity of CONSIM.

CHAPTER 5

QUEUING THEORY

5.1 Queuing and system design

Because there are queues in systems such as ISOS queuing theory finds application in the design of computer systems. In systems, including computer operating systems, new requests may not be served immediately because all the servers are busy. Such waiting requests form queues until the server is available after serving the deserving previous requests. Usually, the request arrival pattern follows a random distribution, often approximately exponential in nature. A system designer attempting to maximize customer satisfaction while minimizing cost would like to know the average value and distribution of parameters such as: queue length, time spent in the queue, server utilization, expectation of preemption and the like. With this information a system can be designed so that response time is satisfactory while the system utilization is sufficiently high.

In this thesis I do not intend to reproduce the common sketches of queuing systems with the many arrangements of servers and multiple different queue structures; rather I would like to point out the values and shortcomings of queuing procedures. There are many good texts and articles

on the application of queuing theory to computer systems. A. K. Erlang carried out the first systematic study of queues during the early twentieth century while studying the Danish telephone network. Since that time a wealth of articles and texts on the subject, as applied to computer systems, has been published. This author feels that competent system designers and analysts should be aware of queuing techniques, what they do and do not do, how to use them, when to use them and how to interpret the results. For a very accurate rendition of the system, simulation is capable of superior results as compared to queuing theory but these good results require a carefully built model and accurate input data. At the early stages of design the input data is not so rigorously defined and frequently the errors are such that simulation is no better than queuing theory.

James Martin (1967) in 'Design of Real-Time computer Systems' has an excellent chapter on Probability and Queuing Theory presented as a 'cookbook' approach for a systems designer having basic albebra. Formulas, tables and curves are presented. Martin advocates the use of simple queuing theory early in system design and always before carrying out simulation. Martin's book leads a reader and designer through exponential and Poisson distributions onwards to single and multiserver queues. He considers queue sizes,

priority and preemptive queues and the probability of queues exceeding different sizes.

A basic application to ISOS would be to answer the question 'what is the probability of needing to interrupt a batch job in order to service an inquiry?' Assume that the batch jobs are all WATFOR and that their execution times are exponentially distributed. As there is equal probability of an inquiry arriving at any time we can assume that, on average, one half of the WATFOR execution remains when the inquiry arrives. Let TR equal the time remaining then we rephrase the question as 'With an exponential service time having mean of M/2 seconds what is the probability of TR IH + TS/2' where IH and TS are the interrupt hold and time-slice times respectively. Again due to the random arrival of inquiries we can assume that, on average, there will be 1/2 a time-slice remaining when the 'need-to-stop' flag is set. Define the question as:

$$P(TR \leq (IH + TS/2)) = 1 - EXP\ (-2(IH + TS/2)/M).$$

If we wish to know what value of IH will require that only 5% of the WATFOR jobs be interrupted then we get:

$$.95 = 1 - EXP\ (-2(IH + TS/2)/M)$$
$$\text{or} \quad \emptyset.5 = EXP\ (-2(IH + TS/2)/M)$$

this leads to

$$.692 = 2(IH + TS/2)/M$$

If we use a TS = Ø.5 seconds to provide maximum service

discipline flexibility as discussed in chapter 3 we get

$$.692 = 2(IH + .25)/1.9$$
$$\text{or} \quad .349 = 2IH$$

Hence from simple queuing theory we see that a first test value of IH of approximately 1/2 second indicates that no more than 5% of the batch jobs need to be interrupted.

As models and systems become more complex or the questions become more detailed then the development of a queuing model will likely require the services of a person whose main interest and education are in the field of queuing theory. On the other hand the literature contains many complex models and there is a chance that an approximation to a given system can be found. Chang (1966) produced a detailed paper on a simple time-sharing system with a feedback queue. In a system with a feedback queue a service request which is not completed at the end of an assigned quantum of processing is returned to the queue to await the assignment of another quantum. It is important to note that the service requests are not serviced to completion in one contiguous amount of processing but rather in a series of seperate small amounts. A second article by Chang (1970) reviewed a number of single server queuing models especially applicable to terminal oriented systems. Chang discussed many aspects of computer systems and tabulated their respective queue type, analysis needed, and

number of servers. This paper is quite complex but is a good example of what to expect in the literature. Other examples of the practical application of queuing techniques are texts by Saul Stimler, 'Real Time Data Processing Systems (1969)' and 'Operating System Principles' by Per Brinch Hansen (1973). A.Ø. Allen (1975) wrote an extensive paper on the application of queuing theory to computing systems. The paper includes the fundamentals of queuing theory and a number of worked out examples.

Hillier and Lieberman (1967) in their book on operations research have a chapter on queuing theory. They include the formulas and analysis for a number of single and multiple server models with various inputs and service distributions. They feel that the emphasis has been on the mathematical descriptive theory and what is needed is research into the proper procedures for using queuing theory. They devote a chapter to applying the techniques and work up to cost models where cost of service is graphed against level of service. They feel that such models help a designer answer important questions regarding the system size and structure.

Martin stressed the use of queuing techniques early in the design phase. Using his approach a reasonable estimate of the system parameters is possible. Unfortunately as the design progresses the system becomes more complex and from

my experience many systems designers are not able to apply queuing theory at this later stage. To apply queuing theory to a complex system may require simplifying assumptions that invalidate the results. Chang (1970) draws attention to this fact when he states,

> 'Although one may calculate second moments in each model - the procedure may not yield sufficiently accurate results because the variance is sensitive to the assumptions made for the simplified model...For a more detailed analysis, simulaton techniques should be used'

Even the complex models require a set of assumptions that do not always reflect the real world. A paper by Kleinrock (1968) on time-shared processors starts with the phrase that indicates the extent of simplifying assumptions:

> 'Assume both that the think time for each console and that the size of each request are exponentially distributed... all quanta are assumed to be infinitesimal, and the swap-time (the time lost in changing jobs) is assumed to be zero.'

Most models assume the exponential interarrival patterns but Walter and Wallace (1967) studied over 10,000 jobs from the University of Michigan Computing Centre and found that the exponential while approximately fitting the observed data was not as good as the hyperexponential curve. The exponential underestimates the long jobs. Coffman and Wood (1966) in their analysis of interarrival statistics in time sharing systems were very critical of the exponential assumption claiming that except as the roughest

approximation it was not justified. They found the biphase hyperexponential to be more satisfactory. The other assumption common in models is the negligible system overhead and swap time. It is well known that such assumptions are not tenable in the real world. Despite the necessary assumptions needed to develop a queuing model the technique is the best way to get an initial feel or estimate for the system. Then as the system form becomes more defined and complex the techniques may need to be abandoned in favour of simulation. Only if accurate answers are needed and warrant the required effort should simulation be attempted.

James Martin states that simulation is useful for heavily used communications lines, especially if polling is used and for assessing the relative merits of supervisor mechanisms, eg. paging schemes, and complex file systems. He feels that it is not worth simulating terminal systems in most cases. He feels that simulation becomes worthwhile if accurate results are required for complex systems with facility utilization of over 70%.

In conclusion then, queuing theory is extremely useful early in the design stage and any analyst working in the systems design area should make queuing theory one of his everyday working tools.

CHAPTER 6

CONCLUSIONS

The conclusions drawn from my thesis research are several, due to the varied aspects and areas into which I delved. The success or failure of the research depends on which point of view is taken and what expectations my supervisor and I had for the work.

6.1 ISOS Summary

The initial aim of my work was to provide an interactive graphics service for more than two hours during the main day shift. The goal was to be achieved without requiring more main execution memory to be dedicated to one service for an extended period of the day. The techniques developed and described in Chapter 3 were capable of fulfilling this goal. Hence one can conclude that the major purpose of this thesis was satisfactorily concluded. However, ISOS was conceived to operate in a particular environment, namely the University of Alberta Computing Centre with the main computer under the control of OS/MVT. Just when the feasability of ISOS was proven the Computing Centre decided to replace OS/MVT by MTS (the Michigan Terminal System). As MTS is an overall time-sharing

operating system a special purpose evolving system such as ISOS was redundant and obsolete. From this point of view the thesis research produced useless results.

On the more positive side one must consider the overall concept of ISOS especially as it evolved from a system to service graphics to a system to service inquiries. In its final form the instance of ISOS was designed to support SOBF and GRID applications but the ISOS idea and design were capable of serving any inquiry programs. A particular feature of inquiry processing was the use of due-date-scheduling of service rather than immediate preemption of the background job. The longer the permissible response time of the IP the lower the ISOS system overhead becomes. The aim of acceptable rather than best possible response time together with good execution memory utilization was achieved in ISOS. From the point of view of the generality of ISOS it can be seen that there was general success and the work was worthwhile.

Despite the lack of a continuing need for ISOS at this university, I feel that the work on the simulation techniques, the queuing theory discussion and the detail on systems design have greatly extended the original aims and the final work might prove useful as a reference for an operating systems design course. The simulation section in particular has explored and tested new techniques. The

actual ISOS model constructed using CONSIM illustrates the validity of the CONSIM approach. From the overall view point then I feel that the thesis research was worthwhile and the final document illustrates the phases of systems design from concept through theory and history onto analysis and evaluation leading to the final fruition of a useful working operating system. The side work on simulation illustrates the development of new techniques to solve a system designer's needs.

6.2 What next?

An instance of ISOS exists, it works and the techniques are correct, but the system has no current direct use. The design techniques are transferrable to other situations by other designers. The simulation of ISOS illustrated the value of the CONSIM approach. CONSIM provides a most fertile area for future research and development. This author intends to convert all the CONSIM functions to IBM's newest APL system, APL/SV. On APL/SV, the new features such as interlocked communicating terminals may prove useful for simulating person to person interactions.

Therefore, as a final summary it must be concluded the ISOS successfully served an educational purpose, explored and developed some operating system design ideas, and opened the door to new techniques for simulation.

BIBLIOGRAPHY

Allen, A.O., 1975. 'Elements of queuing theory for systems design'. I.B.M. Systems Journal 14(2), p.161-187.

Bridges, D.B.J., 1967. 'The executive programs for the LACONIQ time-shared retrieval monitor'. Fall Joint Comput. Conf. p.231-242.

Bryan, G.E., 1967. 'JOSS, 20,000 hours at a console - a statistical summary'. Fall Joint Comput. Conf. p.769-777.

Castleman, Paul A., 1967. 'An evolving special purpose time-sharing system'. Computers and Automation 15(10), Oct. 1967, p.16-18.

Chang, W., 1966. 'A Queuing Model for a Simple Case of Time-Sharing'. I.B.M. System Journal 5(2) p.115-125.

Chang, W., 1970. 'Single Server Queuing Processes in Computer Systems'. I.B.M. Systems Journal 9(1) p.36-71.

Coffman, E.G. and Wood, R.C., 1966. 'Interarrival Statistics for Time-Sharing Systems'. Comm - ACM, 9(7) p.500-503.

Coffman, E.G. Jr., and Kleinrock, L., 1968. 'Computer Scheduling Methods and Their Countermeasures' Spring Joint Comput. Conf. p.11-71.

Darga, K., 1970. 'On-line inquiry under a small system operating system'. I.B.M. Systems Journal 9(1), p.2-11.

Earl, D.B. and Bugely, F.L., 1969. 'Basic Time-Sharing: A System of Computing Principles'. ACM Second Symposium on Operating System Principles, October 1969. p.75-79.

Hansen, Brinch Per, 1973. 'Operating System Principles'. Prentice Hall, Inc. Englewood Cliffs, New Jersey.

Harrop, Michael, 1970. 'A basic approach to remote access'. The Computer Journal 13(2), May 1970, p.131-135.

Hillier, F.S., and Lieberman, G.J., 1967. 'Introduction to Operations Research'. Holden-Day, Inc., San Fransisco.

IBM, 1974. 'APL Shared Variables TSIO Referance Manual'.

IBM, 1973. 'APL/360 User's Manual'.

IBM, 1974. 'APL Shared Variables Users' Guide'.

Kleinrock, Leonard, 1968. 'Some Recent Results for Time Shared Processors'. Proceedings of the Hawaii International Conference on System Science, p.756-759.

Martin, James, 1967. 'Designed Real-Time Computer Systems'. Prentice - Hall, Inc. Englewood Cliffs, New Jersey. P.53 & 372.

McLeod,John, 1970. 'Simulation Today, From Fuzz to Fact'. Simulation Today. No.3,p.9-12

Mealy, G.H., 1966. 'The functional structure of OS/360'. Part 1, IBM Systems Journal, 5(1), p.1-9.

Miller, Robert B., 1968. 'Response time in man computer conversational transactions'. Fall Joint Comput. Conf. Vol. 35, p.267-277.

Needham, R.M. and Hartley, D.F., 1969. 'Theory and Practice In Operating System Design'. ACM Second Symposium on Operating System Principles, October 1969. p.8-12.

Nielsen, N.R., 1967. 'An Approach to the Simulation of a time-sharing system'. Fall Joint Comput. Conf. Vol.31, p.419-428.

Rosin, Robert F., 1969. 'Supervisory and monitor systems'. Computing Surveys 1(1), March 1969, p.37-54.

Sackman, H., 1968. 'Time-sharing versus batch processing: the experimental evidence'. Spring Joint Comput. Conf. Vol. 32, p.1-10.

Sherman, Jack E. and Heying, Douglas W., 1969. 'Performance modelling and empirical measurements in a system designed for batch and time-sharing users'. Fall Joint Comput. Conf. Vol. 35, p.17.

Smyth J.M., 1972. 'YORK APL', Ryerson Polytechnical Institute, Toronto.

Stimler, Saul, 1961. 'Real Time Data-Processing Systems'. McGraw-Hill Inc., New York, New York.

Stimler, S. and Brons, K.E., 1968. 'A methodology for calculating and optimizing real time system performance'. Comm. of A.C.M. 11(7), p.509-516.

Teichroew, D. and Lubin, J.F., 1966. 'Computer Simulation - Discussion of Technique and Comparison of Languages'. Comm of ACM 9(10), p.723-741.

Thomas, E.M., 1967. 'Systems considerations for graphical data processing'. Computers and Automation, 16(11), Nov. 1967, p.17.

Van de Goor, A., Bell, G.C., and Witcraft, D.A., 1969. 'Design and behavior of TSS/8: a PDP-8 based time-sharing system'. IEEE Transactions on Computers, C-18(11), Nov. 1969, p.1038-1043.

Walter, E.S., and Wallace, V.L., 1967. 'Further Analysis of a Computing Center Environment'. Comm of ACM 10(5) p.266-272.

Wood, T.C., 1967. 'A generalized supervisor for a time-shared operating system'. Fall Joint Comput. Conf. p.209-214.

APPENDIX A

INQUIRY SERVICE ORIENTED SUPERVISOR
Master routine, Submonitor A, and Submonitor B
Source Code Listings

```
         MACRO
&NAME    INITIAL &RB,&SA
*              THIS MACRO SAVES REGISTERS IN SAVE AREA &SA AND SETS
*              UP REGISTER # &RB AS THE BASE REGISTER
&NAME    STM   14,12,12(13)
         BALR  &RB,0
         USING *,&RB
         LR    10,13
         LA    13,&SA
         ST    10,4(13)
         ST    13,8(10)
         MEND
         TITLE 'INQUIRY SERVICE ORIENTED SUPERVISOR'
ASJSUPER CSECT
         INITIAL 12,SAVE1
         BC    15,INITERM
         TITLE 'INTERRUPT HANDLER'
**********  INTERRUPT PROCESSOR **********
*              INTERRUPT HANDLER CODE
*              RETURN TO THIS CODE FOLLOWS AN INITIATION OF A BATCH JOB
*              A READ FROM INREADER OR REINITIATION OF AN IP
*              EXIT OCURRS WHEN AN INQUIRY INTERRUPT IS RECIEVED
*              A BATCH JOB ENDS OF A READ IS COMPLETED
INTHAND1 WAIT  1,ECBLIST=INTLIST  WAIT FOR ONE OF THE EVENTS
*              AFTER THE WAIT IS SATISFIED BOTH THE WAIT AND POST BITS
*              OF THE NON POSTED ECBS ARE SET TO 0 WHILE THE POSTED
*              ECB HAS ITS POST BIT SET TO 1
*              DECIDE WHICH EVENT WAS POSTED
*              THERE ARE 3 MAJOR EVENTS 1:READ COMP,2:BATCH END,
*              3:INQUIRY INTERRUPT
         TM    PSG1,X'40'     WAS PSG1 POSTED?
         BC    8,INTHAND2     NO,BRANCH
*                             DECIDE WHO ABENDED/ENDED
         SR    R7,R7
         IC    R7,PSG1+3
         ST    R9,PSG1        ZERO PSG1
         ST    R9,PSA4        PSA4 IS POSTED IF PSG1 IS
         ST    R9,PSB4        PSB4 IS POSTED IF PSG1 IS
         LA    R14,SCHED1     FOR BRANCH IF BATCH ENDED/ABENDED
         BC    15,*+4(R7)
         B     BATCH13        BATCH NORMAL END CODE=0
         B     BATCH13        BATCH ABEND CODE=4
         B     INQIRY14       INQUIRY #1 ABEND CODE=8
INTHAND2 TM    PSB4,X'40'     WAS PSB4 POSTED (INQ OR IP ENDED)
         BC    1,INQIRY13
         TM    INDECB,X'40'   WAS A READ COMPLETE ?
         BC    1,INREAD4      READ COMPLETE
*                             NEW INTERRUPT WAS AN INQUIRY FOR AN IP
         SR    R7,R7
         IC    R7,INQINT+3    GET POST CODE FOR DECODING
         B     INTHAND4
INTHAND3 SR    R7,R7
         IC    R7,INQINT2+3   EXTRACT POST CODE
INTHAND4 ST    R9,INQINT      ZERO THE TWO INTERRUPT ECBS
         ST    R9,INQINT2
         BC    15,*+4(R7)
         B     INQIRY11
*              FOLLOWING THE ABOVE WOULD A BRANCH FOR EACH INQUIRY JOB
INTLIST  DC    A(PSG1)        SET UP THE ECB LIST FOR THE WAIT
         DC    A(INDECB)
         DC    A(PSB4)
         DC    B'10000000'
         DC    AL3(INQINT)
*              END OF THE INTERRUPT HANDLER CODE
*
```

```
         TITLE 'MASTER SCHEDULER'
********* PROGRAM ADMINISTRATOR *********
*              MASTER SCHEDULER
*              THE SCHEDULING OF NEW JOBS ,INTERRUPTED JOBS,AND NEW
*              INQUIRY INTERRUPTS IS DECIDED BY THE USER AND CODED IN
*              THIS ROUTINE
*              THIS VERSION SERVICES PENDING INTERRUPTS , INTERRUPTED
*              BATCH JOBS,AND THEN NEW JOBS (EITHER BATCH OR INQUIRY)
*              ANY PENDING INQUIRY INTERRUPTS
SCHED1   TM    INQINT2,X'40'
         BC    1,INTHAND3
*              ANY INTERRUPTED BATCH JOBS
         TM    STATE+7,X'04'
         BC    1,BATCH12      RESTART THE BATCH JOB
*                             NOTHING PENDING GET NEXT JOB
*              CORE MUST BE FREE BEFORE GETTING THE JOB
         TM    CORSTATE,GOTCORE  TEST THE CORE STATE
         BC    14,INREAD1     CORE IS FREE GET THE NEXT JOB
         BAL   R14,CORE1      BRANCH TO THE ROUTINE TO FREE CORE
         B     INREAD1        GET THE NEXT JOB
*              END OF MASTER SCHEDULER
*
         TITLE 'BATCH SCHEDULER'
********** PROGRAM ADMINISTRATOR *********
*
*                             SCHEDULER OF THE BATCH JOB
*                             NEW BATCH JOB
BATCH10  LA    R7,4(R7)
*                             PLACE REQUEST NAME IN PSA5
         BAL   R14,SCAN2      SCAN FOR REQUESTED SERVICE
         LTR   R15,R15        WAS A REQUEST FOUND
         BNZ   INERROR5       NO,BRANCH TO ERROR ROUTINE
         ST    R7,REQADDR
         LTR   R4,R4          TEST MEMORY STATE
         BZ    BATCH11        SYSTEM IS IDLE THEREFORE NO NEED TO SWAP
*              NOT IDLE  SWAP OUT NEEDED BUT NO SWAP IN NEEDED
         SR    R7,R7          SET NO SWAPIN NEEDED
         BAL   R14,SWAP10
BATCH11  LA    R4,4(0)        SET THE MEMORY TO BATCH ACTIVE
         L     R1,=F'8'
         ST    R1,STATE+4     SET BATCH INDICATOR TO ACTIVE
********** I/O PROCESSOR **********
         CLOSE READER         INPUT DATA SET MUST BE CLOSED BEFORE A
*                             BATCH JOB STARTS
         MVI   INPUT,CLOSEFLG   SET OPEN FLAG=CLOSE
*        POST  PSA5           ENABLE BATCH SUBMONITOR (COMPLETION CODE
*                             IS THE ADDRESS OF THE REQUEST)
         LA    1,PSA5
         L     0,REQADDR      LOAD R0 WITH ADDRESS OF REQUEST
         SVC   2              ISSUE POST SVC
         B     INTHAND1       WAIT FOR AN INTERRUPT
*              INTERRUPTED BATCH JOB
BATCH12  EQU   *
********** PROGRAM ADMINISTRATOR **********
         LA    R7,4(0)        SET UP BATCH SWAP REQUEST
         BAL   R14,SWAP10     EXECUTE NEEDED SWAP
         L     R1,=F'8'
         ST    R1,STATE+4     UPDATE BATCH STATE
         POST  PSA2           RE-ENABLE BATCH SUB MONITOR
         B     INTHAND1
*              JOB END
BATCH13  ST    R9,STATE+4     SET BATCH MONITOR IDLE
         ST    R9,PSA4        CLEAR PSA4 FOR NEXT USE
         ST    R9,PSG1        CLEAR PSG1 FOR NEXT USE
         SR    R4,R4          SET MEMORY STATE = IDLE
         BR    R14            RETURN
REQADDR  DC    1F'0'          ADDRESS OF BATCH REQUEST
*              END OF BATCH SCHEDULER
*
*
```

```
          TITLE 'INQUIRY #1 SCHEDULER'
********** PROGRAM ADMINISTRATOR **********
*
*                 INQUIRY SCHEDULER FOR INQUIRY JOB #1
INQIRY10  TM      STATE+11,X'FF'  CAN NOT HAVE 2 INQUIRY #1 JOBS IN THE
*                                 SYSTEM AT ONCE AS NO QUEUEING FACILITY IS
*                                 INCLUDED IN THE PRESENT VERSION
          BM      ERROR2          SYSTEM CONTAINS AN INQUIRY #1 REJECT
*                                 THE NEW ONE
          LA      R7,0(0)         SET UP INDICATORS FOR SPOOLING ROUTINE
*                                 R7=0 INDICATES INQUIRY #1
          LA      R8,3(0)
          BAL     R14,SPOOLER     BRANCH TO THE ROUTINE TO SPOOL THE JOB
          L       R1,=F'8'
          ST      R1,STATE+8      UPDATE SUBMONITOR#1 STATE
          LA      R4,8(0)         UPDATE MEMORY STATE
          POST    PSB5            ENABLE THE SUB MONITOR
          B       INTHAND1        BRANCH TO INTERRUPT HANDLER
*                 ENTER HERE TO TEST THE MEMORY STATE,IF BATCH ACTIVE
*                 INITIATE INTERRUPTION,IF NOT START SERVICE
INQIRY11  TM      STATE+7,X'08'   TEST BATCH STATE
          BC      8,INQIRY12
          L       1,INQ1HOLD      LOAD REG 1 WITH THE DELAY TIME FOR SERVICE
          BAL     R5,INTRPT1
INQIRY12  LA      R7,8(0)         SET UP REQUEST FOR SWAP ROUTINE
          BAL     R14,SWAP10      GET THE INQUIRY JOB IN CORE
          POST    PSB2            RE-ENABLE THE SUBMONITOR
          L       R1,=F'8'
          ST      R1,STATE+8      UPDATE SUBMONITOR#1 STATE
          B       INTHAND1
*                 INTERRUPT.  3 CAUSES: JOB COMPLETE CODE 4, JOB ABEND
*                 CODE 4, OR ONLY CURRENT INQUIRY COMPLETE CODE 0
INQIRY13  TM      PSB4+3,X'04'    JOB ENDED
          ST      R9,PSG1         ZERO THE TWO ECBS
          ST      R9,PSB4
          BZ      SCHED1          ONLY THE CURRENT INTERRUPT COMPLETE
INQIRY14  SR      R4,R4           SET MEMORY STATE IDLE
          ST      R9,STATE+8      SET SUBMONITOR#1 =IDLE
          B       SCHED1
          DS      0F
INQ1HOLD  DC      X'00000064'     A HOLD TIME OF ONE SEC
*                 END OF INQUIRY #1 SCHEDULER CODE
*
          TITLE 'BATCH INTERRUPT EFFECTOR'
********** PROGRAM ADMINISTRATOR **********
*
*                 INTERRUPT CODE
*                 FOR EFFECTING THE INTERRUPTION OF BATCH JOBS
*                 ON ENTRY R1 CONTAINS THE DELAY TIME FOR
*                 INTERRUPTION INITIATION
INTRPT1   STIMER REAL,INTRPT4,BINTVL=(1) SET UP AN INTERRUPT = HOLD TIME
          WAIT  1,ECBLIST=INTRPTLI WAIT FOR TIMER TO EXPIRE OR BATCH END
          TTIMER CANCEL           CANCEL ANY REMAINING TIMER
          ST    R9,INTRPTEC       ZERO THE TIMER ECB
          TM    PSA4,X'40'        DID THE BATCH JOB END
          BC    3,INTRPT2         BATCH ENDED
          STC   R10,PSA3+3        SET THE NEED TO STOP FLAG=4
          WAIT  1,ECB=PSA4        WAIT FOR SUBMONITOR TO STOP
          MVI   PSA4,X'00'        ZERO THE ECB
INTRPT2   TM    PSA4+3,X'04'      WAS IT JOB END OR INTERRUPT
          BC    1,INTRPT3         BRANCH IF INTERRUPT
          BAL   R14,BATCH13       JOBEND
INTRPT3   BR    R5                RETURN
          USING INTRPT4,R15
INTRPT4   STM   R14,R1,TIMERSAV  SAVE THE REGISTERS CHANGED BY EXIT
          POST  INTRPTEC          INDICATE TIMER EXPIRED
          LM    R14,R1,TIMERSAV
          BR    R14
          DROP  R15
INTRPTEC  DC    1F'0'             ECB FOR STIMER EXPIRATION
INTRPTLI  DC    A(PSA4)           START OF ECB LIST OF THE WAIT
          DC    X'80'
          DC    AL3(INTRPTEC)
*                 END OF THE INTERRUPT EFFECTOR
*
```

```
         TITLE 'CORE MANAGMENT ROUTINES'
********** STORAGE ADMINISTRATOR **********
*              CORE MANAGMENT ROUTINES
*              THE ONLY SUPERVISOR MANAGMENT IS GETMAIN AND FREEMAIN
*              THESE ARE NEEDED FOR SWAPPING
*                              CORE1 ISSUES A FREEMAIN IF GETMAINS ISSUED
CORE1    TM     CORSTATE,GOTCORE HAVE ANY GETMAINS BEEN ISSUED
         BCR    14,R14           NO,RETURN
         FREEMAIN  V,A=DYNAMIC
         MVI    CORSTATE,FREECORE UPDATE CORE STATE
         BCR    15,14            RETURN AFTER FREEING THE CORE
*                                 CORE2 ISSUES A GETMAIN FOR ALL FREE CORE
CORE2    GETMAIN  VC,LA=COREASK,A=COREGOT
         MVI    CORSTATE,GOTCORE
         LTR    R15,R15          DID GETMAIN WORK
*              DUE TO SUBPOOLS ALLOCATING FROM THE TOP OF THE
*              DYNAMIC AREA THE LENGTH NEEDS TO BE CALCULATED
*              AFTER EACH GETMAIN
         L      R1,COREGOT       ADDRESS FROM GETMAIN
         A      R1,COREGOT+4     ADDRESS OF TOP OF CORE
         S      R1,DYNAMIC       ADDRESS OF BOTTOM OF DYNAMIC
         ST     R1,DYNAMIC+4     BOTTOM DYNAMIC TO TOP OF CODE
         BCR    8,R14
         B      ERROR4           FAILURE ON GETMAIN
*                                 CORE3 RESETS PROTECTION KEYS FOR FREE
*                                 SPACE AT TOP OF DYNAMIC AREA
CORE3    C      R9,FREELENG(R4)   ANY FREE CORE
         BCR    8,R14            NO,RETURN
         L      R1,FREEADDR(R4)
         L      R2,FREELENG(R4)
         STM    R1,R2,CORECHG
         FREEMAIN V,A=CORECHG
         BCR    15,R14
COREASK  DC     1F'0'            MINIMUM REQUEST
         DC     1F'150000'       TAKE ALL THE CORE THAT IS FREE
COREGOT  DC     1F'0'            ADDRESS OF ASSIGNED CORE
         DC     1F'0'            LENGTH OF ASSIGNED CORE
CORECHG  DC     2F'0'            ADDR AND LENGTH OF CORE TO BE FREED
FREELENG EQU    *-4
         DC     1F'0'            LENGTH OF FREE CORE FOR BATCH
         DC     1F'0'            LENGTH OF FREE CORE FOR INQUIRY#1
FREEADDR EQU    *-4
         DC     1F'0'            ADDRESS OF FREE CORE FOR BATCH
         DC     1F'0'            ADDRESS OF FREE CORE FOR INQUIRY#1
CODELENG EQU    *-4
         DC     1F'0'            BATCH LENGTH IN BLOCKS
         DC     1F'0'            INQUIRY#1 LENGTH IN BLOCKS
DYNAMIC  DC     2F'0'            ADDRESS AND LENGTH OF DYNAMIC AREA
CORSTATE DC     X'00'            CORE STATE INDICATOR (INITIALLY = FREE)
GOTCORE  EQU    X'FF'            FLAG FOR A GET MAIN ISSUED
FREECORE EQU    X'00'            FLAG FOR A FREEMAIN ISSUED
*              END OF CORE MANAGMENT
*
         TITLE 'SWAP ROUTINES'
********** STORAGE ADMINISTRATOR **********
*                                 ANY SUBMONITOR THAT IS GOING TO DELETE
*                                 A LOAD MODULE MUST FIRST FIND THE ADDRESS
*                                 AND LENGTH OF ANY FREE AREA ABOVE
*                                 THE MODULE
SWAPEP1  STM    14,12,12(13)     SAVE REGS
         LR     R11,R15
         USING  SWAPEP1,R11      EST R11 AS BASE REG
FINDFBQE L      R7,TCBSA         LOAD A SUBTASK TCB ADDRESS
         L      R7,132(0,R7)     LOAD ADDRESS OF PARENT TCB
         L      R8,152(0,R7)
         LA     R8,8(0,R8)       R8 POINTS TO DUMMY PQE
         L      R8,0(0,R8)       R8 POINTS TO THE REAL PQE
         L      R7,0(0,R8)       R7 POINTS TO FIRST FBQE
         CR     R7,R8            WAS THERE A FREE AREA
         BE     SWAPEP1B         NO FREE AREA ; BRANCH
         ST     R7,FBQE          SAVE ADDRESS OF FREE AREA
         L      R1,8(0,R7)       GET THE LENGTH OF FREE AREA
         ST     R1,FBQE+4        SAVE LENGTH
SWAPEP1A LM     R14,R12,12(R13)
         BCR    15,R14
```

```
SWAPEP1B SR     R1,R1
         ST     R1,FBQE         ZERO IN ADDR IF NONE FREE
         ST     R1,FBQE+4       ZERO IF NONE FREE
         BC     15,SWAPEP1A
         DROP   R11
*                                SWAPINIT IS CALLED FROM INITIALIZATION/
*                                TERMINATION CODE TO SET UP SWAP ADDRESSES
*                                AND DISC ADDRESSES OF THE CORE IMAGES
SWAPINT1 BAL    R14,CORE2       SET PROTECTION KEYS FOR SWAP INITIALIZE
         OPEN    (SWAPDISC,(OUTPUT))
         LM     R14,R15,COREGOT
         STM    R14,R15,DYNAMIC  SAVE ADDR AND LENGTH OF DYNAMIC AREA
         LA     R8,4(0,0)
         L      R7,NUMSUBS      LOAD R7 WITH NUMBER OF SUBMONITORS
*                                LOOP INITIALIZES DISC FOR EACH SUBMONITOR
SWAPINT2 LA     R3,1(0,0)       INDICATE ONE BLOCK OUT
         BAL    R5,SWAPOUT1
         NOTE   SWAPDISC
         ST     R1,DISCADDR(R8) DISC ADDRESS OF START OF CORE IMAGES
         L      R3,DYNAMIC+4    LENGTH OF DYNAMIC AREA
         SR     R2,R2
         D      R2,=F'2048'     DETERMINE NUMBER OF BLOCKS IN DYNAMIC
         L      R2,DYNAMIC      ADDRESS OF DYNAMIC AREA
         BAL    R5,SWAPOUT3
         LA     R8,4(0,R8)      INCREMENT R8 BY 4
         BCT    R7,SWAPINT2
         CLOSE   SWAPDISC
         BAL    R14,CORE1       RELEASE THE CORE USED FOR DISC INITIALIZE
         BR     R6
NUMSUBS  DC     1F'2'           NUMBER OF SUBMONITORS IN CURRENT VERSION
*
*               ON ENTRY TO THE SWAP ROUTINES R7 CONTAINS A CODE WHICH
*               INDICATES WHICH SUBMONITOR NEEDS TO BE BROUGHT IN
*               THE REQUIRED JOB MAY BE :IN CORE,OR ON THE DISC:
*               THE CORE MAY CONTAIN: NOTHING OF VALUE,AN INTERRUPTED
*               BATCH JOB,OR AN INQUIRY JOB
*
*               DECIDE CORE CONTENTS AND IF SWAP OUT REQUIRED
SWAP10   ST     R14,SWAPRET     SAVE RETURN ADDRESS
         LTR    R4,R4           IS CORE EMPTY ?
         BZ     SWAP20          CORE IS EMPTY BRING NEXT JOB IN
         CR     R4,R7           DOES CORE CONTENTS=REQUEST
         BCR    8,R14           YES,RETURN
*                                SWAP OUT AND POSSIBLY SWAPIN NEEDED
         OPEN   (SWAPDISC,(OUTIN))
         C      R9,NEW(R4)      IS THE NEW FLAG = NEW I.E. =0
         BC     7,SWAP11        NO,OLD CODE  THEREFORE CALCS NOT NEEDED
         L      R3,FBQE         ADDRESS OF TOP OF CODE
         S      R3,DYNAMIC      ADDRESS OF BOTTOM OF CODE
         SR     R2,R2
         D      R2,=F'2048'     R3 CONTAINS THE NUMBER OF 2K BLOCKS
         ST     R3,CODELENG(R4)
         ST     R10,NEW(R4)     SET NEW FLAG TO OLD I.E. TO 4
         LM     R1,R2,FBQE
         ST     R1,FREEADDR(R4)
         ST     R2,FREELENG(R4)
SWAP11   POINT   SWAPDISC,DISCADDR(R4) POINT TO NEEDED DISC SAVE AREA
         BAL    R14,CORE2       SET DYNAMIC AREA PROTECTION KEYS
         L      R3,CODELENG(R4) LOAD THE NUMBER OF BLOCKS TO BE SWAPPED
         BAL    R5,SWAPOUT1     EXECUTE SWAPOUT
         L      R1,=F'4'
         ST     R1,STATE(R4)    INDICATE INTERRUPTED STATUS
         SR     R4,R4           INDICATE EMPTY CORE
         B      SWAP30
SWAP20   OPEN   (SWAPDISC,(INPUT))  SWAPIN ONLY
         BAL    R14,CORE2       SET DYNAMIC AREA PROTECTION KEYS
SWAP30   LTR    R7,R7           IS A SWAP IN NEEDED
         BC     8,SWAP40        NO,RETURN
         LR     R4,R7           UP DATE CORE CONTENTS
         POINT SWAPDISC,DISCADDR(R4)  POINT DISC FOR NEEDED CORE IMAGES
         L      R3,CODELENG(R4)
         BAL    R5,SWAPIN1      EXECUTE SWAP IN
         L      R1,=F'8'
         ST     R1,STATE(R4)    INDICATE ACTIVE STATUS
         BAL    R14,CORE3       RESET PROTECTION KEYS OF FREE HIGH CORE
```

```
         BC     15,SWAP50
SWAP40   BAL    R14,CORE1       RESET PROTECTION KEYS FOR ALL DYNAMIC AREA
*                                CLOSE THE DATA SET AND RETURN
SWAP50   CLOSE  SWAPDISC
         L      R14,SWAPRET     RETRIEVE RETURN ADDRESS
         BR     R14             RETURN
         DS     0F
NEW      EQU    *-4             NEW FLAGS  =0 IF NEW CODE, =4 IF OLD
         DC     1F'0'           NEW FLAG FOR FIRST SUBMONITOR
         DC     1F'0'           NEW FLAG FOR SECOND SUBMONITOR
*
*                                ON ENTRY TO SWAPOUT1 R3 CONTAINS THE
*                                NUMBER OF BLOCKS TO BE WRITTEN OUT
SWAPOUT1 L      R2,DYNAMIC      ADDRESS OF BOTTOM OF CODE
SWAPOUT2 WRITE  SWAPDECB,SF,,(2),MF=E
         CHECK  SWAPDECB
         ST     R9,SWAPDECB     ZREO THE DECB FOR READ/WRITE
SWAPOUT3 LA     R2,2048(R2)     INCREMENT AREA ADDRESS
         BCT    R3,SWAPOUT2
         BR     R5              RETURN
SWAPIN1  L      R2,DYNAMIC      ADDRESS OF BOTTOM OF CODE
SWAPIN2  READ   SWAPDECB,SF,,(2),MF=E
         CHECK  SWAPDECB
         ST     R9,SWAPDECB     ZREO THE DECB FOR READ/WRITE
         LA     R2,2048(R2)     INCREMENT AREA ADDRESS
         BCT    R3,SWAPIN2      IF MORE BLOCKS TO COME IN GOTO SWAPIN1
         BR     R5              RETURN
*
SWAPRET  DC     1F'0'           RETURN ADDRESS FROM SWAP ROUTINES
DISCADDR EQU    *-4
         DC     1F'0'           DISC ADDRESS OF BATCH CORE IMAGES
         DC     1F'0'           DISC ADDRESS OF INQUIRY #1 CORE IMAGES
         READ   SWAPDECB,SF,SWAPDISC,,2048,MF=L
SWAPDISC DCB    DDNAME=STORSWAP,BLKSIZE=2048,DEVD=DA,DSORG=PSU,BUFNO=0, C
               KEYLEN=0,LRECL=2048,MACRF=(RP,WP),RECFM=F,EODAD=ERROR5
*
*              END OF SWAP ROUTINES
*
         TITLE 'ISOS INITIALIZATION/TERMINATION CODE'
********** FACILITIES AND PROGRAM ADMINISTRATOR **********
*               R4=CORE CONTENTS
*               R4=0   EMPTY
*               R4=4   BATCH
*               R4=8   INQ#1
*
*               SUBMONITOR STATES  0=IDLE  4=INTERUPTED 8=ACTIVE
*              SET THE INTERRUPT EXITS TO OFF
*              BLANK SPIE CLEARS ANY INTERRUPT EXITS
INITERM  SPIE
*              SET COMMON CONSTANTS IN REGISTERS
         SR     R9,R9           A ZERO
         L      R10,=F'4'       A FOUR
*              SET THE CORE CONTENTS REGISTER TO EMPTY(0)
         SR     R4,R4           R4 IS A RESERVED REG FOR CORE CONTENTS
*              CHANGE MASTERS DISPATCH PRIORITY
         CHAP   -1,'S'          REDUCE DP BY 1
*              TASK PRIORITY
*
*              LP=I,DP=I: INQUIRY SUBMONITORS
*              LP=I,DP=I-1: MASTER MONITOR
*              LP=I-2,DP=I-2: BATSH SUBMONITOR
*
*              SET SUBMONITOR STATES TO IDLE I.E. TO ZREO
         ST     R9,STATE+4      BATCH SUBMONITOR STATE
         ST     R9,STATE+8      B SUBMONITOR STATE
*              CREATE THE SUBMONITORS. EACH IS ATTACHED AS A SUB TASK
         LA     R11,ABENDSA     SET UP R11 FOR ADDRESSING IN AGO
         BAL    R14,AGO
         LA     R11,ABENDSB     SET UP R11 FOR ADDRESSING IN BGO
         BAL    R14,BGO
*              THERE WOULD BE A SIMILAR BAL FOR EACH SUB
*                               MONITOR CREATED
*                               ALL SUB MONITORS ARE CREATED AND READY
*                               INITIALIZE THE SWAP ROUTINE
         BAL    R6,SWAPINT1     INITIALIZE THE SWAP ROUTINES
```

```
*                                  GOTO TO INPUT READER TO RETRIEVE FIRST JOB
          B      INREAD1
*                                  TERMINATION OF ISOS
NOJOBS    TM     STATE+11,X'FF'    IS INQUIRY #1 IDLE
          ST     R9,INDECB         ZERO THE READ DECB
          BC     7,INTHAND1        INQUIRY #1 NOT IDLE AWAIT AN INTERRUPT
          ABEND  1112              ALL SUB MONITORS IDLE AND INPUT EMPTY
*
          TITLE 'ABEND/ATTACH ROUTINES FOR SUBMONITORS'
*
*                 SUBMONITOR ABEND AND ATTACH ROUTINES
*                 THIS CODE IS ENTERED TO INITIALLY CREATE THE SUBMONITORS
*                 AND TO DETACH AND RE-ATTACH THEM WHEN THEY ABEND
ABENDSA   STM    14,12,12(13)      START CODE FOR SUB MONITOR A (BATCH)
          LR     R11,R15           EST BASE REGISTER
          USING  ABENDSA,R11
          LR     R10,R13           SET UP SAVE AREAS AND CHAIN THEM
          LA     R13,ETXRSAV
          ST     R10,4(R13)
          ST     R13,8(R10)
          CLI    ATTACHA,X'FF'     DID ABEND OCCUR DURING AN ATTACH
          BNE    ABENDSA1          NO,THEREFORE CONTINUE
          ABEND  1115,DUMP         ABEND AND FIND CAUSE
*                ON ABEND OF SUBTASK A ENTER HERE FROM S/360
ABENDSA1  DETACH TCBSA             DETACH THE TCB THAT ABENDED
          BAL    R14,AGO           RE-ATTACH
          L      R1,ADDRPSA4
          POST   (1),4             NOTIFICATION OF SUBMONITOR A ABEND
          L      R1,ADDAPSG1
          POST   (1),4             NOTIFY INTRRUPT HANDLER OF ABEND
          L      R13,4(R13)
          RETURN (14,12)
*                ATTACH SUBMONITOR A
AGO       MVI    ATTACHA,X'FF'     SET THE ATTACH FLAG ON
          L      R7,ADDRPSA1
          SR     R1,R1             A ZERO FOR NEXT INST. AND FOR TCB ADDR
          ST     R1,0(0,R7)        ZERO THE ECB AT PSA1
          ATTACH EP=SUBA,LPMOD=2,DPMOD=-1,ETXR=ABENDSA,PARAM=(PSA1,PSA2,C
                PSA3,PSA4,PSA5,PSG1,SWAPEP1)
          ST     R1,TCBSA
          WAIT   1,ECB=PSA1        WAIT FOR NOTIFICATION THAT SUB A IS READY
          MVI    ATTACHA,X'00'     SET ATTACH FLAG OFF
          BR     R14
ADDRPSA1  DC     A(PSA1)
ADDRPSA4  DC     A(PSA4)
ADDAPSG1  DC     A(PSG1)
ATTACHA   DC     X'00'             ATTACH FLAG
          DROP   R11
ABENDSB   STM    14,12,12(13)      START CODE FOR SUB MONITOR B (INQUIRY #1)
          LR     R11,R15           EST BASE REGISTER
          USING  ABENDSB,R11
          LR     R10,R13           SET UP SAVE AREAS AND CHAIN THEM
          LA     R13,ETXRSAV
          ST     R10,4(R13)
          ST     R13,8(R10)
          CLI    ATTACHB,X'FF'     DID ABEND OCCUR DURING AN ATTACH
          BNE    ABENDSB1          NO,THEREFORE CONTINUE
          ABEND  1115,DUMP         ABEND AND FIND CAUSE
*                ON ABEND OF SUBTASK B ENTER HERE FROM S/360
ABENDSB1  DETACH TCBSB             DETACH THE TCB THAT ABENDED
          BAL    R14,BGO           RE-ATTACH
          L      R1,ADDRPSB4
          POST   (1),4             NOTIFICATION OF SUBMONITOR B ABEND
          L      R1,ADDBPSG1
          POST   (1),8             NOTIFY INTRRUPT HANDLER OF ABEND
          L      R13,4(R13)
          RETURN (14,12)
*                ATTACH SUBMONITOR B
BGO       MVI    ATTACHB,X'FF'     SET THE ATTACH FLAG ON
          L      R7,ADDRPSB1
          SR     R1,R1             A ZERO FOR NEXT INST. AND FOR TCB ADDR
          ST     R1,0(0,R7)        ZERO THE ECB AT PSB1
          ATTACH EP=SUBB,LPMOD=0,DPMOD=+1,ETXR=ABENDSB,PARAM=(PSB1,PSB2,C
                PSB3,PSB4,PSB5,PSG1,INQINI,INQINT2,SWAPEP1)
          ST     R1,TCBSB
```

```
         WAIT  1,ECB=PSB1    WAIT FOR NOTIFICATION THAT SUB B IS READY
         MVI   ATTACHB,X'00' SET ATTACH FLAG OFF
         BR    R14
ADDRPSB1 DC    A(PSB1)
ADDRPSB4 DC    A(PSB4)
ADDBPSG1 DC    A(PSG1)
ATTACHB  DC    X'00'         ATTACH FLAG
         DROP  R11
*              THERE WOULD BE A SIMILAR ROUTINE FOR EACH SUB MONITOR
*              END OF ABEND/ATTACH ROUTINES
         TITLE 'INPUT READER'
********** I/O PROCESSOR **********
*              INPUT READER CODE
*              THE FIRST CARD IS RETRIEVED USING A 'READ' IN CASE
*              THERE IS A WAIT FOR THE NEXT JOB.
*              IS THE INPUT DATA SET OPEN? IF NOT THEN OPEN IT
INREAD1  CLI   INPUT,OPENFLAG
         BE    INREAD3       DATA SET IS OPEN
         OPEN  (READER,(INPUT)) OPEN THE INPUT DATA SET
         LA    R3,READER     ADDR TO CHECK THE OPEN
         USING IHADCB,R3
         TM    DCBOFLGS,X'10' TEST BIT 3 FOR SUCCESSFUL OPEN
         DROP  R3
         BZ    ERROR1        OPEN FAILED
         MVI   INPUT,OPENFLAG SET THE OPEN FLAG=OPEN
INREAD3  READ  INDECB,SF,MF=E
         B     INTHAND1      AFTER ISSUING THE READ GO TO THE
*              INTERRUPT HANDLER TO AWAIT EITHER THE END OF THE
*              READ OR AN INQUIRY INTERRUPT
*              READ COMPLETE TEST ITS SUCCESS
INREAD4  TM    INDECB,X'0F'  DID READ COMPLETE NORMALLY
         BC    14,ERROR3     NO,BRANCH TO ERROR ROUTINES
         ST    R9,INDECB     ZERO THE ECB FOR NEXT USE
*              DECODE THE 'JOB' AND 'EXEC' CARDS
INREAD2  LA    R11,MSGAREA   ADDR FOR INPUT,COMMON AREA WITH LIST
         CLC   0(2,R11),=CL2'$$'  IS IT A $$ CARD
         BNE   INREAD3       IF NOT IGNORE AND GET NEXT CARD
*              TO ALLOW FOR SOME ERROR IN 'JOB' AND 'EXEC' CARDS A
*              SCAN ROUTINE LOOKS FOR THE NEEDED DATA, BLANKS ARE USED
*              AS DELIMITERS. HOPEFULLY THIS REDUCES THE NUMBER OF
*              REJECTED JOBS DUE TO SLIGHT 'JCL TYPE' ERRORS
         LA    R7,2(R11)     START OF CARD SCAN
         LA    R8,70(R11)    END OF CARD SCAN
         BAL   R14,SCAN2     BRANCH TO SCAN FIND NONBLANK AFTER BLANK
         LTR   R15,R15       IS RETURN CODE =0, I.E. SUCCESS
         BNZ   INREAD3       NO,GET THE NEXT CARD
*                            JOB CARD FORMAT:  $$*** JOB *****
         CLC   0(3,R7),=CL3'JOB'  IS IT A JOB CARD
         BNE   INREAD3       NOT A 'JOB' CARD  GET NEXT CARD
*                            'JOB' CARD FOUND  LIST IT IF DESIRED
         LA    R1,84(0)      SET UP MESSAGE LENGTH
         BAL   R14,LIST      LOG THE MESSAGE
*              LOOK FOR 'EXEC' CARD AND REQUESTED FACILITY
         READ  INDECB,SF,MF=E
         CHECK INDECB
         ST    R9,INDECB
         CLC   0(2,R11),=CL2'$$'  IS IT A $$ CARD
         BNE   INERROR1
         LA    R7,2(R11)
         LA    R8,70(R11)
         BAL   R14,SCAN2     SCAN FOR THE WORD EXEC
         LTR   R15,R15       IS RETURN CODE =0, I.E. SUCCESS
         BNZ   INERROR2      NO NON-BLANK ON THE CARD
         CLC   0(4,R7),=CL4'EXEC'  IS THE NON BLANK 'EXEC'
         BNE   INERROR3
         LA    R7,4(R7)
********** FACILITIES ADMINISTRATOR **********
         BAL   R14,SCAN2     LOOK FOR THE REQUESTED FACILITY
         LTR   R15,R15       IS RETURN CODE =0, I.E. SUCCESS
         BNZ   INERROR4      NO FACILITY REQUESTED
         CLC   0(4,R7),=CL4'SOBF'  IS SOBF THE REQUESTED FACILITY
         BE    BATCH10
*              THE INQUIRY CODE BLOCKS ARE NUMBERED 10,20,30...ETC
*              THE SEQUENCE OF  CLC/BE  WOULD BE REPEATED FOR EACH
*              FACILITY OFFERRED
```

```
         CLC   0(4,R7),=CL4'GRID'  IS GRID THE REQUESTED FACILITY
         BE    INQIRY10
         B     INERROR4       THE REQUESTED FACILITY IS NOT SUPPORTED
*        INPUT READER ERROR ROUTINES
*              SECOND CARD HAS NO $$
INERROR1 LA    R7,ERRNOTE1
         B     INERROR
*              ON A SEARCH FOR NON BLANKS ONLY BLANKS FOUND
INERROR2 LA    R7,ERRNOTE2
         B     INERROR
*                            'EXEC' NOT FOUND ON SECOND CARD
INERROR3 LA    R7,ERRNOTE3
         B     INERROR
*              NON SUPPORTED FACILITY REQUESTED
INERROR4 LA    R7,ERRNOTE4
         B     INERROR
INERROR5 LA    R7,ERRNOTE5
         B     INERROR
INERROR  MVC   MSGAREA+80(18),0(R7)  MOVE IN MESSAGE
         LA    R1,102(0)       SET UP MESSAGE LENGTH
         BAL   R14,LIST        LOG THE TOTAL MESSAGE
         B     INREAD1         LOOK FOR THE NEXT JOB
ERRNOTE1 DC    CL18'NO $$ ON 2ND CARD'
ERRNOTE2 DC    CL18'NON BLANKS MISSING'
ERRNOTE3 DC    CL18'EXEC NOT 2ND WORD'
ERRNOTE4 DC    CL18'NO/INVALID FACILITY'
ERRNOTE5 DC    CL18'NO SERVICE REQUEST'
INPUT    DC    X'00'           STATE OF INPUT DATA SET (INITIALLY CLOSED)
OPENFLAG EQU   X'FF'
CLOSEFLG EQU   X'00'
         READ  INDECB,SF,READER,MSGAREA,80,MF=L
READER   DCB   DDNAME=FT05F001,DSORG=PS,LRECL=80,BLKSIZE=80,RECFM=F,   C
               EODAD=NOJOBS,MACRF=(R),DEVD=DA,BUFNO=1
*              END OF INPUT READER ROUTINE
*
         TITLE 'SCAN ROUTINE'
*              SCAN ROUTINE
*              ON ENTRY R7 CONTAINS THE STARTING ADDRESS FOR SCAN
*              AND R8 THE LIMIT ADDRESS.        SCAN LOOKS FOR THE
*              1ST NONBLANK AFTER A BLANK AND RETURNS THE ADDRESS IN R7
*              ENTER AT SCAN1 IF AN INCREMENT NEEDED BEFORE SCAN START
*              ENTER AT SCAN2 IF LOOKING FOR A BLANK THEN A NONBLANK
*              ENTER AT SCAN3 IF INITIAL INCREMENT NEEDED AND LOOKING
*              FOR NONBLANK.  R15=0 IF SUCCESS AND=4 IF FAILURE.
SCAN1    LA    R7,1(R7)        START OF BLANK SEARCH
         CR    R7,R8           CAN SCAN CONTINUE
         BH    SCAN4           LIMIT ADDRESS OF SCAN REACHED  STOP SCAN
SCAN2    CLI   0(R7),X'40'  LOOK FOR A BLANK
         BNE   SCAN1           NOT A BLANK THEREFORE LOOK AGAIN
SCAN3    LA    R7,1(R7)        START NON BLANK SEARCH
         CR    R7,R8           CHECK LIMIT
         BH    SCAN4           STOP  LIMIT REACHED
         CLI   0(R7),X'40'  IS IT A BLANK
         BE    SCAN3           YES  LOOK AGAIN
         LA    R15,0           SET RETURN CODE FOR NON BLANK FOUND
         BR    R14
SCAN4    LA    R15,4           SET RETURN FOR LIMIT REACHED DURING SCAN
         BR    R14
*              END OF SCAN ROUTINE
*
*
         TITLE 'LIST ROUTINE'
*              LIST ROUTINE
*              THIS ROUTINE UTILIZES AN EXECUTE FORM OF A WTL
LIST     STH   R1,MESSAGE      R1 CONTAINS THE LENGTH+4
         WTL   MF=(E,MESSAGE)
         BR    R14             RETURN
MESSAGE  WTL   ' ',MF=L
MSGAREA  EQU   *-1
         DS    CL100           SET UP THE ACTUAL MESSAGE AREA
*              END OF LIST ROUTINE
*
```

```
         TITLE 'SPOOLING ROUTINE'
********** I/O PROCESSOR **********
*
*                              START OF GENERAL SPOOLING ROUTINE
*                              ON ENTRY R7 INDICATES WHO TO SPOOL AND
*                              R8 CONTAINS THE MAXIMUM NUMBER OF FILES
*                              TO BE SPOOLED
*                              AN R7 = 0 INDICATES INQUIRY #1
SPOOLER  STM    14,13,SUBSAVE
         LA     R3,SPOOL
         USING  IHADCB,R3
         L      R7,FILENAMS(R7)
         BAL    R14,SETDCB     SET UP THE DCB TO HANDLE NEXT DATA SET
READ1    READ   INDECB,SF,MF=E
         CHECK  INDECB
         ST     R9,INDECB
         CLC    0(1,R11),=CL2'$$'  IS IT A $ CARD
         BNE    READ1          GARBAGE OR $ CARD MISSING TRY NEXT
NEXTFILE CLC    1(4,R11),0(R7) DECIDE FILE NAME
         BNE    EMPTY          THE EXPECTED DATA SET NOT NAMED
*                THERE IS POSSIBLY DATA FOR THE NEXT DATA SET
*                REWIND THE DATA SET AND WRITE OUT THE CARD IMAGES
         BAL    R14,CLOSER
         BAL    R14,OPENFILE
AGAIN    READ   INDECB,SF,MF=E
         CHECK  INDECB
         ST     R9,INDECB
         CLC    0(1,R11),=CL2'$$'  IS IT A $ CARD
         BE     ENDFILE        NEXT DATA SET STARTING OR END OF LAST ONE
         PUT    SPOOL,(R11)    CARD WRITTEN TO APPROPRIATE DATA SET
         B      AGAIN
CLOSER   CLOSE  (SPOOL,REREAD) CLOSE THE DATA SET WITH REREAD
         BR     R14
ENDFILE  BAL    R14,CLOSER     CLOSE DATA SET WITH A REREAD
         B      EMPTY2
EMPTY    CLOSE  (SPOOL,LEAVE)  EMPTY DATA SET CLOSE WITH A LEAVE
EMPTY2   LA     R7,8(R7)       INCREMENT POINTER FOR DATA SET NAMES
         BCT    R8,*+8         ANY MORE DATA SETS TO CHECK
         B      SPOOLEND       ALL DATA SETS PROCCESSED
         BAL    R14,SETDCB     MORE DATA SETS   RESET THE DCB
         B      NEXTFILE
SPOOLEND LM     14,13,SUBSAVE
         BR     R14
SETDCB   LM     R0,R1,0(R7)    GET NAME OF NEXT FILE
         STM    R0,R1,DCBDDNAM  INSERT NAME IN DDNAME FIELD OF DCB
OPENFILE OPEN   (SPOOL,(OUTPUT))
         BR     R14
FILES1   DC     CL8'FORTRAN '  INQUIRY #1 (GRAPHICS) FILE NAMES
         DC     CL8'OBJECT  '
         DC     CL8'FT03F001'
FILENAMS EQU    *
INQ1FILE DC     A(FILES1)
*                              THERE WOULD BE AN ADDRESS DC FOR EACH JOB
*                              REQUIRING SPOOLING AND A DD CARD FOR EACH
*                              FILE NAME LISTED
SUBSAVE  DC     16F'0'
SPOOL    DCB    BLKSIZE=80,DDNAME=FORTRAN,DEVD=DA,DSORG=PS,LRECL=80,    C
               MACRF=PM,RECFM=F,BUFNO=1
*              END OF SPOOLING ROUTINE
*
         TITLE 'GENERAL ERROR ROUTINES'
*              GENERAL ERROR ROUTINES
*              ALL ERROR MESSAGES (ERRMSG) ARE OF LENGTH 30
ERROR1   LA     R7,ERRMSG1
         B      ERRORPT1
ERROR2   LA     R7,ERRMSG2
         LA     R8,INREAD1     SET UP THE RETURN ADDR FOR THIS ERROR
         B      ERRORPT2
ERROR3   TM     INDECB,X'01'   DID THE DATA SET END?
         BC     1,NOJOBS       YES,GO TO END OF DATA SET ROUTINE
         LA     R7,ERRMSG3     INPUT READ FAIL
         B      ERRORPT1
ERROR4   LA     R7,ERRMSG4
         B      ERRORPT1
ERROR5   LA     R7,ERRMSG5
```

```
          B      ERRORPT1
ERRMSG1   DC     CL30'OPEN FAILED ON INPUT'
ERRMSG2   DC     CL30'INQUIRY#1 ALREADY IN SYSTEM'
ERRMSG3   DC     CL30'INPUT READ FAILED'
ERRMSG4   DC     CL30'GET MAIN IN CORE MGNT FAILED'
ERRMSG5   DC     CL30'END OF DATA DURING A SWAP'
ERRORPT1  SR     R8,R8            INDICATE NO RETURN AFTER ERROR PROCESSED
ERRORPT2  MVC    ERRORPT3+25(30),0(R7)  MOVE IN THE ERROR MESSAGE
ERRORPT3  WTL    'GENERAL ERROR                                    '
          LTR    R8,R8            DECIDE NEXT ACTION (ABEND OR RETURN)
          BCR    7,R8             NON ZERO THEREFORE RETURN
ERRORPT4  ABEND  2000,DUMP        NONRECOVERABLE ERROR  ABEND SUPERVISOR
*
*
*                 END OF ERROR ROUTINES
*
          TITLE  'MISCELANEOUS DEFINITIONS'
TCBSA     DC     1F'0'            TCB ADDRESS SAVE AREA
TCBSB     DC     1F'0'            TCB ADDRESS SAVE AREA
SAVE1     DC     18F'0'
ETXRSAV   DC     18F'0'           A SAVE AREA FOR ABEND ROUTINES
INQINT    DC     1F'0'            PRIMARY INQUIRY INTERRUPT ECB
INQINT2   DC     1F'0'            SECONDARY INQUIRY INTERRUPT ECB
PSA1      DC     1F'0'
PSA2      DC     1F'0'
PSA3      DC     1F'0'
PSA4      DC     1F'0'
PSA5      DC     1F'0'
PSB1      DC     1F'0'
PSB2      DC     1F'0'
PSB3      DC     1F'0'
PSB4      DC     1F'0'
PSB5      DC     1F'0'
PSG1      DC     1F'0'
TIMERSAV  DC     4F'0'            A SAVE AREA FOR TIMER EXITS
FBQE      DC     2F'0'            FREECORE PARAMETER SAVE AREA
STATE     EQU    *-4
          DC     1F'0'            BATCH STATE WORD
          DC     1F'0'            INQUIRY#1 STATE WORD
R0        EQU    0
R1        EQU    1
R2        EQU    2
R3        EQU    3
R4        EQU    4
R5        EQU    5
R6        EQU    6
R7        EQU    7
R8        EQU    8
R9        EQU    9
R10       EQU    10
R11       EQU    11
R12       EQU    12
R13       EQU    13
R14       EQU    14
R15       EQU    15
          DCBD   DSORG=PS,DEVD=DA      SET UP DUMMY DCB FOR SPOOL
          END
```

DSTAT RESPONSE

SAMPLE SIZE	7
MAXIMUM	10.07605081
MINIMUM	5.867687184
RANGE	4.208363623
MEAN	7.416275818
VARIANCE	2.930301502
STANDARD DEVIATION	1.711812344
MEAN DEVIATION	1.412428335
MEDIAN	6.556191142
MODE	

DSTAT BREQ

SAMPLE SIZE	11
MAXIMUM	18.61670137
MINIMUM	1.281136935
RANGE	17.33556443
MEAN	8.836912956
VARIANCE	44.52769082
STANDARD DEVIATION	6.672907224
MEAN DEVIATION	5.375520324
MEDIAN	8.130704191
MODE	

DSTAT BTIME

SAMPLE SIZE	6
MAXIMUM	95.63509354
MINIMUM	3.283757932
RANGE	92.3513356
MEAN	65.12247353
VARIANCE	1067.321365
STANDARD DEVIATION	32.66988468
MEAN DEVIATION	22.53231906
MEDIAN	75.25135741
MODE	

SUBMONITOR A

```
         MACRO
&NAME    PURGE &LIST           LIST ADDR
&NAME    IHBINNRA  &LIST       LOAD REG 1
         SVC   16              ISSUE SVC FOR PURGE
         MEND
         MACRO
&NAME    RESTORE  &LIST
&NAME    IHBINNRA  &LIST       LOAD REG 1
         SVC   17              ISSUE SVC FOR RESTORE
         MEND
         MACRO
&NAME    INITIAL &RB,&SA
.*             THIS MACRO SAVES REGISTERS IN SAVE AREA &SA AND SETS
.*             UP REGISTER # &RB AS THE BASE REGISTER
&NAME    STM   14,12,12(13)
         BALR  &RB,0
         USING *,&RB
         LR    10,13
         LA    13,&SA
         ST    10,4(13)
         ST    13,8(10)
         MEND
MONA     CSECT
         TITLE 'SUBMONITOR A (BATCH) CODE'
*
SUBA     INITIAL 12,SAVESA1
         LM    3,9,0(1)          LOAD REGS WITH ADDRS OF THE PARMS
*                                PSA#      1    2    3    4    5    6
*                                REG #     3    4    5    6    7    8
*                                R9 CONTAINS THE ADDR OF SWAPEP1
*                                R2 CONTAINS COMPILER STATE
*                                0=LOADED  4=NOT LOADED
         ST    R9,SWAPADA
         STM   4,7,PSAS          PASS PSA 2,3,4,5 ADDRESSES TO TXR
         SR    R10,R10           A ZERO
*                                POST ECB AT PSA1
         POST  (3),0             BATCH SUBMONITOR READY TO ACCEPT JOBS
*                                WAIT ON ECB AT PSA5 (NEW JOB)
********** FACILITIES ADMINISTRATOR **********
NEXTJOBA WAIT  1,ECB=(7)         PSA5
*                                DECODE THE SERVICE
         LA    R2,4              SET COMPILER STATE TO NOT LOADED
         LA    R9,SERVICES-8
         L     R1,NUMOPTS
         L     R11,0(0,R7)       THE PSA5 COMPLETION CODE WAS REQUEST ADDR
         ST    R10,0(0,R7)       ZERO THE ECB FOR NEXT USE
CHECK    LA    R9,8(R9)          INCREMENT THE POINTER TO THE SERVICES
         CLC   0(4,R11),0(R9)    COMPARE REQUEST TO OFFERRINGS
         BE    LOADCOMP
         BCT   R1,CHECK          IF MORE OPTIONS LEFT CHECK THEM
         B     JOBENDA2
LOADCOMP LM    R0,R1,0(R9)
         STM   R0,R1,COMPILER
         LOAD  EPLOC=COMPILER
         ST    R0,ENTRY
         LA    R2,0              SET THE COMPILER STATE TO LOAD EXECUTED
         LA    R11,TXR           FOR ADDRESSING
         BAL   14,RESET
         B     JOBSTART
********** TIMER ADMINISTRATOR **********
TXR      STM   14,12,12(13)       START OF STIMER EXIT ROUTINE
         LR    R11,R15           EST R11 AS A BASE REGISTER
         USING TXR,R11
         LR    10,13
         LA    13,SAVESA2
         ST    10,4(13)
         ST    13,8(10)
         LM    4,7,PSAS          LOAD PSA ADDRESSES (2 3 4 5)
         SR    R10,R10
         IC    R10,3(0,R5)       EXTRACT THE NEED TO STOP FLAG
         BC    15,*+4(R10)       IF =0 CONTINUE ;  IF=4 STOP
         BC    15,CONTINUE
```

```
         ST    9,0(0,5)  CLEAR OUT THE SUBA NEED TO STOP FLAG
         PURGE UPPLIST
         LTR   R2,R2          CHECK STATE OF LOAD
         BC    7,TXR2
         L     R15,SWAPADA
         BALR  R14,R15
         BAL   R14,COMPDEL
TXR2     SPIE
         ST    R1,OLDPICAA    SAVE OLD PICA ADDRESS
         POST  (6),4          POST HAVE STOPPED ECB I.E. PSA4
*                             WAIT FOR RESTART
         WAIT  1,ECB=(4)      PSA 2
         ST    R9,0(0,4)      ZERO THE ECB
         RESTORE  INITLINK
         L     R1,OLDPICAA    LOAD OLD PICA ADDRESS
         LTR   R1,R1          WAS THERE AN OLD PICA
         BC    8,CONTINUE     NO,SKIP RESTORE
         SPIE  MF=(E,(1))     RESTORE OLD PICA
CONTINUE BAL   14,RESET
         L     13,4(13)
         RETURN (14,12)
RESET    STM   14,1,12(13)
         STIMER TASK,TXR,BINTVL=SUBAONE
         RETURN  (14,1)
         DROP R11
PSAS     DC    4F'0'          ADDRESSES OF PSA 2 3 4 5
JOBSTART L     R15,ENTRY
         BALR  14,15
********** FACILITIES ADMINISTRATOR **********
JOBENDA1 TTIMER  CANCEL       CANCEL THE TIME SLICER
*                             CLEAR OUT ANY INTERRUPT EXIT ADDRESSES
         SPIE
         LTR   R2,R2          IF COMPILER LOADED DELETE ELSE JOBEND
         BC    7,JOBENDA2
         BAL   R14,COMPDEL
JOBENDA2 POST  (6),0       POST HAVE STOPPED IN CASE WE HAD REQUEST
         POST  (8),0          POST JOB COMPLETE ECB (PSG1) 0=FINISHED
         B     NEXTJOBA
*                             DELETE A COMPILER IF IT HAS BEEN LOADED
COMPDEL  DELETE EPLOC=COMPILER
         LA    R2,4(0,0)      SET R2 TO NO LOAD
         BR    R14
SAVESA2  DC    18F'0'
SAVESA1  DC    18F'0'
ENTRY    DC    1F'0'     MAIN STORAGE ADDR OF LOAD MODULE ENTRY POINT
         DS    0F
SUBAONE  DC    X'00000032'    ONE HALF SECOND SUBMONITOR 'A' TIME SLICE
SWAPADA  DC    1F'0'          ADDR OF SWAPEP1 ROUTINE FINDS CORE BOUNDS
NUMOPTS  DC    1F'5'          NUMBER OF SERVICES OFFERED
COMPILER DC    2F'0'          NEEDED MODULE NAME
SERVICES DC    CL8'WATFOR  '
         DC    CL8'SALT    '
         DC    CL8'PUNCH   '
         DC    CL8'PRINT   '
         DC    CL8'PRTPCH  '
INITLINK DC    1F'0'          POINTER TO INITIAL IOB FOR RESTORE
OLDPICAA DC    1F'0'          SAVE OF ADDRESS OF OLD PICA
         DS    0F             ALLIGN UPPLIST ON FULLWORD
UPPLIST  DC    BL1'00000110'    OPTIONS FOR PURGE
         DC    AL3(0)         DEB ADDR  (OMITTED)
PCCODE   DC    AL1(0)         COMPLETION CODE
TCBADDR  DC    AL3(0)         TCB ADDR
         DC    AL1(0)         QUIECES INDICATOR
         DC    AL3(INITLINK)    ADDRESS OF INITIAL IOB LINK
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
```

```
R8      EQU   8
R9      EQU   9
R10     EQU   10
R11     EQU   11
R12     EQU   12
R13     EQU   13
R14     EQU   14
R15     EQU   15
        END
```

SUBMONITOR B

```
         MACRO
&NAME    INITIAL &RB,&SA
.*             THIS MACRO SAVES REGISTERS IN SAVE AREA &SA AND SETS
.*             UP REGISTER # &RB AS THE BASE REGISTER
&NAME    STM    14,12,12(13)
         BALR   &RB,0
         USING *,&RB
         LR     10,13
         LA     13,&SA
         ST     10,4(13)
         ST     13,8(10)
         MEND
MONB     CSECT

         TITLE  'SUBMONITOR B (INQUIRY #1)'

*                              SUBMONITOR 'B',INQUIRY #1,CURRENTLY
*                              DESIGN TO SERVICE A SIMULATED GRAPHICS
*                              TERMINAL.
*                              INPUT CAN INCLUDE FORTRAN IV(G) SOURCE
*                              CODE,OBJECT CODE, AND DATA SETS.
SUBB     INITIAL 12,SAVESB1
         LM     3,11,0(1)       LOAD REGS WITH ECB ADDRESSES
*                              PSB#       1   2   3   4   5   G   IN  IN2
*                              REG#       3   4   5   6   7   8   9   10
*                              R11 CONTAINS ADDR OF SWAPEP1
         ST     R11,SWAPADB
         IDENTIFY  EP=TIQREAD,ENTRY=TIQBEAD1
         C      R15,=F'0'     DID IDENTIFY WORK
         BNE    IDERROR    DID NOT WORK  ABEND
         LA     15,LOAD
*                              STORE ECB ADDRESSES NEEDED IN INQUIRY EXIT
         ST     R4,4(0,R15)     PSB2
         ST     R6,8(0,R15)     PSB4
         ST     R9,12(0,R15)    IN
         ST     R10,16(0,R15)   IN2
         POST   (3),0           POST  PSB1
********** FACILITIES ADMINISTRATOR **********
NEXTJOBB LR     R1,R7           PSB5
         WAIT   1,ECB=(7)       WAIT FOR POSTING OF PSB5
         MVI    0(R7),X'00'     ZERO PSB5
*                              COMPILE THE SOURCE CODE
         LINK   EP=IEYFORT,PARAM=(OPADDR,NEWDD),VL=1
         C      15,=F'4'        TEST FOR SUCCESSFUL COMPILE
         BC     11,JOBENDB1
*                              INVOKE THE LINKAGE EDITOR
         LINK   EP=IEWL,PARAM=(PARMADDR,DDADDR),VL=1
         C      15,=F'4'        TEST FOR SUCCESSFUL LINK-EDIT
         BC     11,JOBENDB1
         LOAD   EP=INQUIRY      LOAD THE EXECUTABLE LOAD MODULE
         MVI    LOAD,X'FF'      SET THE LOAD FLAG
         LR     R15,R0
         BALR    R14,R15
*              JOB END ROUTINE
*                              CLEAR OUT ANY INTERRUPT EXITS
JOBENDB1 SPIE
         CLI    LOAD,X'FF'      WAS LOAD ISSUED
         BNE    JOBENDB2        NO JOBEND
         BAL    R14,DELETELD    YES DELETE
JOBENDB2 POST   (6),4           INDICATE NORMAL COMPLETION OF SUBB JOB
         B      NEXTJOBB        GO WAIT FOR NEXT JOB
IDERROR  ABEND 1113,DUMP
         DS     0F
OPADDR   DC     X'0000'
         DS     0F
NEWDD    DC     AL2(LENC)
CNAMES   DC     CL8'FORTCOMP'
         DC     BL24'0'
         DC     CL8'FORTRAN '
LENC     EQU    *-CNAMES
```

```
          DS    0F
PARMADDR  DC    AL2(LENL)
OPTIONS   DC    C'XREF,LET,LIST'
LENL      EQU   *-OPTIONS
          DS    0F
DDADDR    DC    AL2(LEND)
LNAMES    DC    CL8'FSYSLIN '
          DC    BL8'0'
          DC    CL8'STEPLIB '
          DC    CL8'FSYSLIB '
          DC    BL8'0'
          DC    CL8'LKEDPRNT'
          DC    BL8'0'
          DC    CL8'FSYSUT1 '
LEND      EQU   *-LNAMES
*                               EXIT FOR THE STIMER USED TO SIMULATE
*                               INTERRUPTS. WOULD NOT BE IN FINAL VERSION
          USING INTEXIT,R15
INTEXIT   POST  INTECB
          BCR   15,R14
          DROP  R15
*                               INQUIRY INTERRUPT ENDED ROUTINE
*                               IN TIQREAD R4=PSB2,R5=PSB4,R6=INQINT,AND
*                               R7=INQINT2
TIQREAD1  STM   14,12,12(13)
          LR    R11,R15         EST R11 AS BASE REGISTER
          USING TIQREAD1,R11
          LR    10,13
          LA    13,SAVEB2
          ST    10,4(13)
          ST    13,8(10)
          CLI   LOAD,X'FF'      WAS A LOAD INITIATED?
          BNE   TIQREAD2
          L     R15,SWAPADB
          BALR  R14,R15
DELETELD  DELETE EP=INQUIRY
          MVI   LOAD,X'00'      SET LOAD FLAG OFF
*                               CLEAR OUT ANY INTERRUPT EXITS
TIQREAD2  SPIE
********** INTERRUPT PROCESSOR **********
          ST    R1,OLDPICAB     SAVE OLD PICA ADDRESS
          POST  (5),0  INTERRUPT SERVICED
*                               GENERATE A DUMMY INTERRUPT DELAY
          STIMER  REAL,INTEXIT,BINTVL=INITIME
TIQREAD4  WAIT  1,ECB=INTECB    WAIT FOR INQUIRY INTERRUPT
          MVI   INTECB,X'00'    ZERO THE INTERRUPT ECB
          POST  (7)             POST INQINT2
          POST  (6)             POST INQINT
          WAIT  1,ECB=(4)   WAIT FOR ISCS TO SAY  GET GOING (PSB2)
          MVI   0(R4),X'00'     ZERO ECB AT PSB2
          L     R1,OLDPICAB     RETRIEVE ADDRESS OF OLD PICA ,IF ANY
          LTR   R1,R1           WAS THERE AN OLD PICA
          BC    8,TIQREAD3      NO OLD PICA,SKIP RESTORE
          SPIE  MF=(E,(1))      RESTORE OLD PICA
TIQREAD3  L     13,SAVEB2+4
          LM  14,12,12(13)
          BCR   15,14
          DROP  R11
INTECB    DC    1F'0'           INQUIRY ECB
INTTIME   DC    1F'0'           DELAY FOR NEXT DUMMY INTERRUPT
OLDPICAB  DC    1F'0'           SAVE OF ADDRESS OF OLD PICA
LOAD      DC    1F'0'           CARD #1 OF A CONTIGUOUS SET
SPSB2     DC    1F'0'           #2
SPSB4     DC    1F'0'           #3
SINQINT   DC    1F'0'           #4
SINQINT2  DC    1F'0'           #5  (LAST OF SET)
SWAPADB   DC    1F'0'           ADDR OF SWAPEP1 ROUTINE FINDS CORE BOUNDS
SAVEB2    DC    18F'0'
SAVESB1   DC    18F'0'
R0        EQU   0
R1        EQU   1
R2        EQU   2
R3        EQU   3
R4        EQU   4
R5        EQU   5
```

```
R6          EQU   6
R7          EQU   7
R8          EQU   8
R9          EQU   9
R10         EQU   10
R11         EQU   11
R12         EQU   12
R13         EQU   13
R14         EQU   14
R15         EQU   15
            END
```

APPENDIX B

CONSIM COMMANDS

```
   BRANCH NEXTESR
[1]ESR←NEXTESR
[2]BR←1


   CONSIMLD
[1]LIB←',1491'
[2]⍞')LOAD ENTINDEX',LIB
[3]⍞')LOAD ENTADD',LIB
[4]⍞')LOAD INITIAL',LIB
[5]⍞')LOAD STOP',LIB
[6]⍞')LOAD SETTRACE',LIB
[7]⍞')LOAD INSERT',LIB
[8]⍞')LOAD ESRADD',LIB
[9]⍞')LOAD NXTEVENT',LIB
[10]⍞')LOAD SETSTAT',LIB
[11]⍞')LOAD EXTRACT',LIB
[12]⍞')LOAD FUTURE',LIB
[13]⍞')LOAD ESRINDEX',LIB
[14]⍞')LOAD DESTROY',LIB
[15]⍞')LOAD BRANCH',LIB
[16]⍞')LOAD CREATE',LIB
[17]⍞')LOAD SIMULATE',LIB
[18]⍞')LOAD DQ',LIB
[19]⍞')LOAD FIFOQ',LIB
[20]⍞')LOAD PRTYQ',LIB
[21]⍞')LOAD TRANSFER',LIB
[22]⍞')LOAD SIGNAL',LIB
[23]⍞')LOAD PREEMPT',LIB
[24]⍞')LOAD RETURN',LIB
[25]⍞')LOAD PASS',LIB
[26]⍞')LOAD EXPON',LIB
[27]⍞')LOAD QOK',LIB
[28]⍞')LOAD QADD',LIB
[29]⍞')LOAD EXECCPU',LIB
```

```
    TYPE_ESR CREATE PARMS;B;TYPE;CESR;WHEN;N;S
[1]B←TYPE_ESR⍳' '
[2]TYPE←(B-1)↑TYPE_ESR
[3]CESR←B↓TYPE_ESR
[4]WHEN←TIME+1↑PARMS
[5]PARMS←(6⍴0),1↓PARMS
[6]PARMS[1]←ENTNUM←ENTNUM+1
[7]N←ESRINDEX CESR
[8]PARMS[3]←N
[9]FIX1:N←ENTINDEX TYPE
[10]→EXISTS2×⍳0≠⍴N
[11](¯6+⍴PARMS) ENTADD TYPE
[12]TYPE,'  ADDED TO ENTITY NAMES'
[13]N←ENTINDEX TYPE
[14]EXISTS2:PARMS[2]←N[1]
[15]→ERR1×⍳N[2]≠⍴PARMS
[16]→ERR2×⍳WHEN<TIME
[17]FIX2:PARMS[5,6]←TIME,WHEN
[18]S←CURRENT
[19]CURRENT←PARMS
[20]FUTURE
[21]⍎TYPE,'←',TYPE,',CURRENT'
[22]CURRENT←S
[23]→0
[24]ERR1:'FOR EXISTING TYPE ',TYPE,' NUMBER OF PARMS IS
[25]STOP 'CREATE'                                  WRONG'
[26]→FIX1
[27]ERR2:'EVENT TIME < CLOCK TIME'
[28]STOP 'CREATE'
[29]→FIX2


    DESTROY;ID;ENT;C;R
[1]→0×⍳CURRENT[1]=0
[2]ID←CURRENT[1,2]
[3]ENT←ENTNAMES[ID[2];]
[4]⍎'C←',ENT,'[1;]⍳ID[1]'
[5]⍎'R←⍴',ENT
[6]OK:⍎ENT,'←((R[1],C-1)↑',ENT,'),(0,C)↓',ENT
[7]CURRENT[4]←DEST


    Z←DQ NAME;R
[1]⍎'R←(⍴',NAME,')[1]'
[2]⍎'Z←,(R,1)↑',NAME
[2.5]→6×⍳0=+/Z
[3]⍎NAME,'←(0,1)↓',NAME
[4]Z SETSTAT PROCESS
[5]→0
[6]'ATTEMPT TO DQ FROM AN EMPTY QUEUE ',NAME
[7]STOP 'DQ'
```

```
    Z←EXPON M
[1]Z←-M×⍟?(⍴M)⍴0
```

```
    ID FIFOQ NAME
[1]→OK×⍳QOK NAME
[2]2 QADD NAME
[3]OK:⍎'NAME,''←'',NAME,'',ID'
[4]→NOT×⍳ID[1]≠CURRENT[1]
[5]CURRENT[4]←QUED
[6]INSERT
[7]→0
[8]NOT:ID SETSTAT QUED
```

```
    Z←MAXFREE
[1]Z←(⌊/TIMELINE[1;])-TIME
```

```
    PASS TOESR
[1]CURRENT[6]←TIME
[2]CURRENT[3]←ESRINDEX TOESR
[3]FUTURE
[4]INSERT
```

```
    IDP PRTYQ NAME
[1]→OK×⍳QOK NAME
[2]3 QADD NAME
[3]OK:⍎'Q←',NAME
[4]→NOT×⍳IDP[1]≠CURRENT[1]
[5]I←+/Q[3;]>CURRENT[IDP[3]]
[6]Q←((3,I)↑Q),((3,1)⍴CURRENT[1,2,IDP[3]]),(0,I)↓Q
[7]CURRENT[4]←QUED
[8]INSERT
[9]⍎NAME,'←Q'
[10]→0
[11]NOT:P←EXTRACT IDP[1 2]
[12]P←P[IDP[3]]
[13]I←+/Q[3;]>P
[14]Q←((3,I)↑Q),((3,1)⍴IDP[1,2],P),(0,I)↓Q
[15]IDP[1,2] SETSTAT QUED
[16]⍎NAME,'←Q'
```

```
    S QADD NAME
[1]QNAMES←QNAMES,[1]8↑NAME
[2]⍎NAME,'←(S,0)⍴0'
```

```
    Z←QOK NAME
[1]NAME←8↑NAME
[2]R←(⍴QNAMES)[1]
[3]Z←R≥(+/QNAMES=(R,8)⍴NAME)⍳8


    ID SETSTAT S
[1]ENT←ENTNAMES[ID[2];]
[2]⍎'C←',ENT,'[1;]⍳ID[1]'
[3]⍎ENT,'[4;C]←S'


    SETTRACE F
[1]TRACE←F


    DELAY SIGNAL ESRN
[1]('DUMMY ',ESRN) CREATE DELAY


    STOP NAME
[1]'SIMULATION HALTED BY CALL TO STOP FROM ';NAME
[2]'SIMULATION RESUMED'


    TRANSFER TOESR
[1]ESR←TOESR
[2]BR←2
```

```
    CPUEND
[1]DESTROY
[2]CPU←IDLE
[3]CURRENT←NCPU
[4]CURRENT[4]←PROCESS
[5]NEXTESR←ESRNAMES[CURRENT[3];]
[6]→OK×ι2≠+/CURRENT[1,2]=0
[7]NEXTESR←ι0
[8]0 SIGNAL ESRNAMES[CURRENT[3];]
[9]OK:NEXTTIME←TIME
[10]→0×ι0=(ρCPUREQ)[2]
[11]NCPU←EXTRACT DQ'CPUQ'
[12]CPU←BUSY
[13](EXECTIME CPUREQ[;1]) SIGNAL 'CPUEND'
[14]CPUREQ←(0 1)↓CPUREQ


    ESRN EXECCPU INT
[1]CURRENT[3]←ESRINDEX ESRN
[2]CURRENT[4]←4
[3]→NO×ιCPU=BUSY
[4]CPU←BUSY
[5]NCPU←CURRENT
[6](EXECTIME INT) SIGNAL 'CPUEND'
[7]→0
[8]NO:CURRENT[1,2] FIFOQ 'CPUQ'
[9]CPUREQ←CPUREQ,(2,1)ρINT,0


    PREEMPT ID;P
[1]P←TIMELINE[2;]ιID[1]
[2]TIMELINE[2 3;P] SETSTAT 6
[3]TIMELINE[1;P]←1E7+TIME-TIMELINE[1;P]


    RETURN ID;P
[1]P←TIMELINE[2;]ιID[1]
[2]TIMELINE[2 3;P] SETSTAT 5
[3]TIMELINE[1;P]←1E7+TIME-TIMELINE[1;P]
```

```
    R ENTADD NAME
[1]ENTNAMES←ENTNAMES,[1]NAME←8↑NAME
[2]⍎NAME,'←((R+6),0)⍴0'


    Z←ENTINDEX NAME;R;EM
[1]NAME←8↑NAME
[2]R←(⍴ENTNAMES)[1]
[3]EM←(R,8)⍴NAME
[4]Z←(+/ENTNAMES=EM)⍳8
[5]→0∘Z←⍳0,→OK×⍳Z≤R
[6]OK:⍎'R←(⍴',NAME,')[1]'
[7]Z←Z,R


    ESRADD NAME
[1]ESRNAMES←ESRNAMES,[1]8↑NAME


    Z←ESRINDEX NAME;R;EM
[1]NAME←8↑NAME
[2]R←(⍴ESRNAMES)[1]
[3]EM←(R,8)⍴NAME
[4]Z←(+/ESRNAMES=EM)⍳8
[5]→0×⍳0≠⍴Z←(Z≤R)/Z
[6]ESRNAMES←ESRNAMES,[1]8↑NAME
[7]→EXIT×⍳~TRACE
[8]NAME,' ADDED TO ESRNAME LIST'
[9]EXIT:Z←R+1


    Z←EXTRACT ID;C
[1]ENT←ENTNAMES[ID[2];]
[2]⍎'C←',ENT,'[1;]⍳ID[1]'
[3]S1:⍎'Z←',ENT,'[;C]'


    FUTURE
[1]TIMELINE←TIMELINE,(3,1)⍴CURRENT[6,1,2]
[2]CURRENT[4]←5


    INSERT
[1]ENT←ENTNAMES[(CURRENT[2]);]
[2]⍎'C←',ENT,'[1;]⍳CURRENT[1]'
[3]S1:⍎ENT,'[;C]←CURRENT'
```

```
    NEXT←NXTEVENT;N;P;T
[1]→MORE×⍳0≠(⍴TIMELINE)[2]
[2]'TIMELINE EMPTY'
[3]STOP 'NXTEVENT'
[4]MORE:TIME←⌊/TIMELINE[1;]
[5]P←TIMELINE[1;]⍳TIME
[6]N←TIMELINE[2;P]
[7]T←TIMELINE[3;P]
[8]TIMELINE←((3,P-1)↑TIMELINE),(0,P)↓TIMELINE
[9](N,T) SETSTAT 0
[10]NEXT←EXTRACT N,T
[11]ESR←ESRNAMES[(NEXT[3]);]


    SIMULATE MAXTIME
[1]INITIAL
[2]GENERATE
[3]NEXT:CURRENT←NXTEVENT
[4]→STOPT×⍳MAXTIME<TIME
[5]RECALL:⍎'∇SIMULATE[11]CALL:',ESR,'∇'
[6]→NT×⍳~TRACE
[7]'      TRACE: ';ESR;' ID ';CURRENT[1,2];' TIME ';TIME
[8]NT:NEXTESR←ESR←⍳0
[9]NEXTSTAT←BR←0
[10]NEXTTIME←⌊/⍳0
[11]CALL:SCHED
[12]→B1×⍳1=BR
[13]→RECALL×⍳2=BR
[14]→NEXT×⍳0≠CURRENT[4]
[15]→DES×⍳0=⍴NEXTESR
[16]CURRENT[3,6]←(ESRINDEX NEXTESR),NEXTTIME
[17]FUTURE
[18]INSERT
[19]→NEXT
[20]STOPT:'SIMULATION STOPPED AS MAXIMUM TIME REACHED'
[21]→0
[22]B1:→B2×⍳CURRENT[4]≠0
[23]DES:DESTROY
[24]→NEXT×⍳1≠BR
[25]B2:CURRENT←6⍴0
[26]→RECALL
```

APPENDIX C

ISOS MODEL

```
   Z←EXECTIME INT
[1]SL←(INT,0)[2]
[2]→((SL=0),(INT[1]<SL),(INT[1]≥SL))/S1,S1,S2
[3]S1:RUN←INT[1]
[3.5]Z←RUN
[4]→0
[5]S2:RUN←⌊/INT[1],SL×⌈((⌈/MAXFREE,SL)÷SL
[5.5]Z←RUN


   GENERATE
[1]NEEDCPU←0
[2]READER←CORE←CPU←0
[3]STATE←0,0
[4]READ←NINQ←SOBEND←SOBABEND←IPABEND←IPEND←INQSERV←0
[5]CPUREQ←2 0⍴0
[6]READQ←(2,0)⍴0
[7]GI←⎕∘⎕←'INQUIRY INTERARRIVAL TIME'
[8]GCL←⎕∘⎕←'GRID COMPILE/LKED TIME'
[9]GS←⎕∘⎕←'INQ SERVICE TIME'
[10]SJH←⎕∘⎕←'SOBF JOBS PER HOUR'
[11]GIR←⎕∘⎕←'INQUIRY JOB INTERARRIVAL TIME'
[12]SJT←⎕∘⎕←'SOBF JOB TIME'
[13]TRACE←'Y'=⍞∘⎕←'SET TRACE ON (Y OR N)'
[14]SWAPTIME←⎕∘⎕←'SWAPTIME '
[15]TS←⎕∘⎕←'TIME SLICE'
[16]IH←⎕∘⎕←'IH'
[17]NEEDCPU←0
[18]NEEDSTOP←0
[19]RESPONSE←BREQ←BTIME←BINT←⍳0
[20]0 SIGNAL 'ISSUERD'
[21]0 SIGNAL 'NEWSOBF'
[22]0 SIGNAL 'NEWGRID'
[23]'INITIALIZATION COMPLETE'


   GRID
[2]→OK×⍳STATE[2]=0
[3]'GRID BUSY ';TIME
[4]BRANCH 'ISSUERD'
[5]→0
[6]OK:'SPOOLEND' EXECCPU ⎕←?6
```

```
    GRIDGO
[2]DESTROY
[3]CURRENT←IPC
[4]CURRENT[4]←PROCESS
[4.5]RESPONSE←RESPONSE,TIME-INQTIME
[5]'INQDONE' EXECCPU □←EXPON GS


    INITIAL
[1]SΔSTOP←2
[2]TIMELINE←3 0ρι0
[3]ENTNUM←0
[4]TIME←0
[5]ESRNAMES←0 8ρ' '
[6]NON:ENTNAMES←0 8ρ' '
[7]PROCESS←0
[8]QUED←1
[9]DEST←2
[10]STORED←3
[11]NONE←ι0
[12]YES←1
[13]NO←0
[14]CURRENT←6ρ0
[15]OFF←0
[16]ON←1
[17]BUSY←1
[18]IDLE←0
[19]QNAMES←0 8ρ' '


    INQAGAIN
[2]NINQ←1
[3]CURRENT[4]←4
[4]IPC←CURRENT
[5]0 SIGNAL 'INTHAND'


    INQDONE
[1]CURRENT[13]←CURRENT[13]+1
[2]→DONE×ιCURRENT[13]=CURRENT[11]
[3]INQSERV←1
[4]□←NEXTTIME←TIME+EXPON GI
[5]NEXTESR←'INQAGAIN'
[6]0 SIGNAL 'INTHAND'
[7]→0
[8]DONE:IPEND←1
[9]0 SIGNAL 'INTHAND'
```

```
   INQWAIT
[1]NEEDCPU←OFF
[2]→BEND×⍳1=SOBEND∨SOBABEND
[3]→0×⍳IHSET=NO
[4]IHSET←OFF
[5]NEEDSTOP←ON
[6]→0
[7]BEND:IHSET←NO
[7.5]SOBEND←SOBABEND←0
[8]BRANCH 'INQ3'


   INQ1
[1]→NO×⍳STATE[1]≠8
[2]IHSET←YES
[3]NEEDCPU←ON
[4]IH SIGNAL 'INQWAIT'
[5]→0
[6]NO:BRANCH 'INQ2'


   INQ2
[1]2 SWAP 'INQ2B'


   INQ2B
[1]0 SIGNAL 'GRIDGO'
[2]STATE[2]←8
[3]BRANCH'INTHAND'


   INQ3
[1]NEEDSTOP←NO
[2]→STOP×⍳(STATE[1]=8)∧SOBEND=0
[3]SOBEND←STATE[1]←CORE←0
[4]STOP:BRANCH 'INQ2'
```

```
    INTHAND
[1]→(SOBEND,SOBABEND,IPABEND,READ,NINQ,INQSERV,IPEND)/
[2]⍝NOTHING COMPLETE                    R3,R4,R5,R1,R2,R7,R6
[3]→0
[4]R1:READ←0
[5]'READ COMPLETE'
[6]CURRENT←EXTRACT DQ 'READQ'
[7]NEXTESR←'JOBCD'
[8]NEXTTIME←TIME+.1
[9]→0
[10]R2:NINQ←0
[11]'NEW INQUIRY'
[11.5]INQTIME←TIME
[12]NEXTESR←'INQ1'
[13]NEXTTIME←TIME+.1
[14]→0
[15]R3:SOBEND←0
[16]'SOBF END'
[17]STATE[1]←CORE←0
[18]BRANCH'SCHED'
[19]→0
[20]R4:'SOBABENDED AT ';TIME
[21]SOBABEND←0
[22]→R3
[23]R5:'IPABENDED AT ';TIME
[24]IPABEND←0
[25]R6:IPEND←0
[26]'IP END'
[27]STATE[2]←CORE←0
[28]BRANCH'SCHED'
[29]→0
[30]R7:INQSERV←0
[31]'INQUIRY SERVED'
[32]BRANCH'SCHED'


    ISSUERD
[2]READ←READ∨(⍴READQ)[2]>0
[3]READER←~READ
[4]BRANCH'INTHAND'
```

```
   JOBCD
[2]→OK×⍳CURRENT[8]=YES
[3]'BAD JOB CARD ';TIME
[4]BRANCH'ISSUERD'
[5]→0
[6]OK:→OK2×⍳CURRENT[9]=YES
[7]'BAD EXEC CARD ';TIME
[8]BRANCH'ISSUERD'
[9]→0
[10]OK2:→S×⍳CURRENT[10]=1
[11]→G×⍳CURRENT[10]=2
[12]'INVALID FACILITY REQUEST ';TIME
[13]BRANCH 'ISSUERD'
[14]→0
[15]S:NEXTESR←'SOBF'
[16]NEXTTIME←TIME+.2
[17]→0
[18]G:NEXTESR←'GRID'
[19]NEXTTIME←TIME+.2


   NEWBAT
[1]'TSEND' EXECCPU ⌊/TS,CURRENT[11]


   NEWGRID
[1]P←0,0,(1≠?50),(1≠?50);2,(⌊GIR÷GI),(1≠?10),0
[2]'GRIDJ READCOMP' CREATE P
[3](EXPON GIR) SIGNAL 'NEWGRID'
[4]'NEW GRID JOB PARMS ';P


   NEWIP
[2]'NEWIP2' EXECCPU ⎕←EXPON GCL


   NEWIP2
[2]→FAIL×⍳CURRENT[12]=0
[3]⍝COMPILE LKED OK
[4]'INQDONE' EXECCPU ⎕←EXPON GS
[5]→0
[6]FAIL: 'GRID COMPILE/LKED FAIL'
[6.5]IPABEND←1
[7]0 SIGNAL 'INTHAND'
```

```
   NEWSOBF
[1]P←0,0,(1≠?20),(1≠?20),1,(EXPON SJT),(1≠?10)
[2]'SOBFJ READCOMP' CREATE P
[3]BREQ←BREQ,P[6]
[4](EXPON 3600÷SJH) SIGNAL 'NEWSOBF'
[5]'NEW SOB JOB PARMS ';P


   READCOMP
[1]CURRENT[1,2] FIFOQ 'READQ'
[2]→0×⍳READER=OFF
[3]READ←1
[4]READER←OFF
[5]BRANCH'INTHAND'


   RESUME
[1]DESTROY
[2]CURRENT←BCUR
[3]CURRENT[4]←0
[4]'TSEND' EXECCPU □←CURRENT[11],TS


   SCHED
[2]→NON×⍳NINQ=0
[3]NINQ←0
[4]CURRENT←EXTRACT DQ'INQQ'
[5]TRANSFER'INQ1'
[6]→0
[7]NON:→0×⍳(NEEDCPU=YES)∨NEEDSTOP=YES
[7.5]→MORE×⍳STATE[1]=4
[8]BRANCH'ISSUERD'
[9]→0
[10]MORE:1 SWAP 'SCHED2'


   SCHED2
[1]CORE←4
[2]STATE[1]←8
[3]0 SIGNAL 'RESUME'
[4]BRANCH'INTHAND'


   SOBF
[2]→NOSERV×⍳0=CURRENT[12]
[2.5]STATE[1]←8
[3]0 SWAP 'SOBF2'
[4]→0
[5]NOSERV:'SOBF ABEND OR NO REQUEST ';TIME
[6]BRANCH 'ISSUERD'
```

```
    SOBF2
[1]S1:CORE←4
[3]PASS'NEWBAT'
[4]BRANCH'INTHAND'


    SPOOLEND
[1]STATE[2]←CORE←8
[2]PASS'NEWIP'
[3]BRANCH'INTHAND'


    N SWAP NESR
[1]P←2,1
[2]→CALC×⍳0=N
[3]STATE[N]←8
[4]→CALC×⍳STATE[P[N]]≠8
[5]STATE[P[N]]←4
[6]CALC:→Z×⍳CORE=N
[7]→Z2×⍳(CORE=0)∨N=0
[8]T←2
[9]→SET
[10]Z:T←0
[11]→SET
[12]Z2:T←1
[13]SET:CORE←N
[14]NESR EXECCPU .1+T×SWAPTIME


    TSEND
[1]CURRENT[11]←CURRENT[11]-RUN
[2]→DONE×⍳CURRENT[11]≤0
[3]→S1×⍳NEEDSTOP=ON
[4]OK:'TSEND' EXECCPU □←CURRENT[11],TS
[5]→0
[6]DONE:SOBEND←1
[7]BTIME←BTIME,TIME-CURRENT[5]
[8]BINT←BINT,CURRENT[7]
[9]→S1×⍳NEEDSTOP=YES
[10]→S2×⍳NEEDCPU=YES
[11]0 SIGNAL 'INTHAND'
[12]→0
[13]S2:0 SIGNAL 'INQWAIT'
[14]→0
[15]S1:0 SIGNAL 'INQ3'
[16]CURRENT[7]←CURRENT[7]+1
[17]CURRENT[4]←4
[18]BCUR←CURRENT
```

```
      SIMULATE 1000
INQUIRY INTERARRIVAL TIME
 □     15
GRID COMPILE/LKED TIME
 □     5
INQ SERVICE TIME
 □     2
SOBF JOBS PER HOUR
 □     200
INQUIRY JOB INTERARRIVAL TIME
 □     300
SOBF JOB TIME
 □     10
SET TRACE ON (Y OR N)
Y
SWAPTIME
 □     2.5
TIME SLICE
 □     .5
IH
 □     1
ISSUERD  ADDED TO ESRNAME LIST
DUMMY  ADDED TO ENTITY NAMES
NEWSOBF  ADDED TO ESRNAME LIST
 NEWGRID  ADDED TO ESRNAME LIST
 INITIALIZATION COMPLETE
      TRACE: ISSUERD  ID 1   1 TIME 0
      TRACE: INTHAND ID 0   0 TIME 0
      TRACE: NEWSOBF  ID 2   1 TIME 0

NEW SOB JOB PARMS 0   0   1   1   1   8.130704191   1
     TRACE: READCOMP ID 75   2 TIME 72.69418895
     TRACE: CPUEND   ID 74   1 TIME 73.41535505
     TRACE: INQ2B    ID 77   1 TIME 73.41535505
     TRACE: INTHAND ID 0   0 TIME 73.41535505
     TRACE: GRIDGO   ID 78   1 TIME 73.41535505
  1.229395238
     TRACE: CPUEND   ID 79   1 TIME 74.64475033
     TRACE: INQDONE  ID 6   3 TIME 74.64475033
  82.29326455
      )SI
 SIMULATE [11]
*INQDONE  [5]
      SΔINQDONE←⍳0
      →5
     TRACE: INTHAND  ID 80   1 TIME 74.64475033
INQUIRY SERVED
     TRACE: SCHED ID 0   0 TIME 74.64475033
     TRACE: NEWSOBF  ID 76   1 TIME 78.1061428
NEW SOB JOB PARMS 0   0   1   1   1   3.951302041   0
     TRACE: READCOMP ID 82   2 TIME 78.1061428
     TRACE: CPUEND   ID 81   1 TIME 79.74475033
```

B30130